DOI 10.3233/FI-2018-1700

A Unified Approach to Multisource Data Analyses

Franck Ravat Jiefu Song*

IRIT - Université Toulouse I Capitole

2 Rue du Doyen Gabriel Marty F-31042 Toulouse Cedex 09,

France ravat@irit.fr, song@irit.fr

Abstract. Classically, *Data Warehouse*s (DWs) supports business analyses on data coming from the inside of an organization. Nevertheless, *Lined Open Data* (LOD) might sensibly complete these business analyses by providing complementary perspectives during a decision-making pro-cess. In this paper, we propose a conceptual modeling solution, named *Unified Cube*, which blends together multidimensional data from DWs and LOD datasets without materializing them in a stationary repository. We complete the conceptual modeling with an implementation frame-work which manages the relations between a *Unified Cube* and multiple data sources at both schema and instance levels. We also propose an analysis processing process which queries different sources in a transparent way to decision-makers. The practical value of our proposal is illustrated through real-world data and benchmarks.

Keywords: Data Warehouse, Linked Open Data, Conceptual Modeling, Multisource analyses, Experimental Assessments

1. Introduction

Well-informed and effective decision-making relies on appropriate data for business analyses. Data are considered appropriate if they include enough information to provide an overall perspective to decision-makers. To obtain as many appropriate data as possible, decision-makers must have access to the company's business data at any time. Since the 1990s, *Business Intelligence* (BI) has been

*Address for correspondence: IRIT - Université Toulouse I Capitole, 2 Rue du Doyen Gabriel Marty F-31042 Toulouse Cedex 09, France

providing methods, techniques and tools to collect, extract and analyze business data stored in a *Data Warehouse* (DW) [9]. However, an overall perspective during decision-making requires not only business data from inside a company but also other data from outside a company. In today's constantly evolving business context, one promising approach consists of blending web data with warehoused data [32]. The concept of BI 2.0 is introduced to envision a new generation of BI enhanced by web-based content [39].

Among various web-based content, *Linked Open Data* (LOD)[1] provide a set of inter-connected and machine-readable data to enhance business analyses on a web scale [45]. Since data are produced and updated at a high speed nowadays, materializing all data (e.g., warehoused data and LOD) related to analyses in one stationary repository can hardly be synchronized with changes in data sources. It is necessary to unify data from various sources without integrating all data into a stationary repository. To support up-to-date decision-making, business dashboards must be created in an on-demand manner. Such dashboards should include all appropriate data required by decision-makers.

Case Study. In a government organization managing social housings, internal data are periodically extracted, transformed and loaded in a DW. As shown in figure 1(a), the DW describes number of applications (i.e. *Applications*) according to two analysis axes: one about the geographical location of social housings (i.e. *Housing_Ward* and *Housing_District*) and the other related to applicant's profile (i.e. *Applicant_Status*). This DW only gives a partial view on the demand for social housings. To support effective decision-making, additional information should be included in analyses. Therefore, a decision-maker browses in a second dataset, named LOD1, to obtain complementary views on social housing allocation. Published by the *UK Department for Communities and Local Government*[2], LOD1 describes the accepted applications for social housing (i.e. *acceptance*) according to *district*

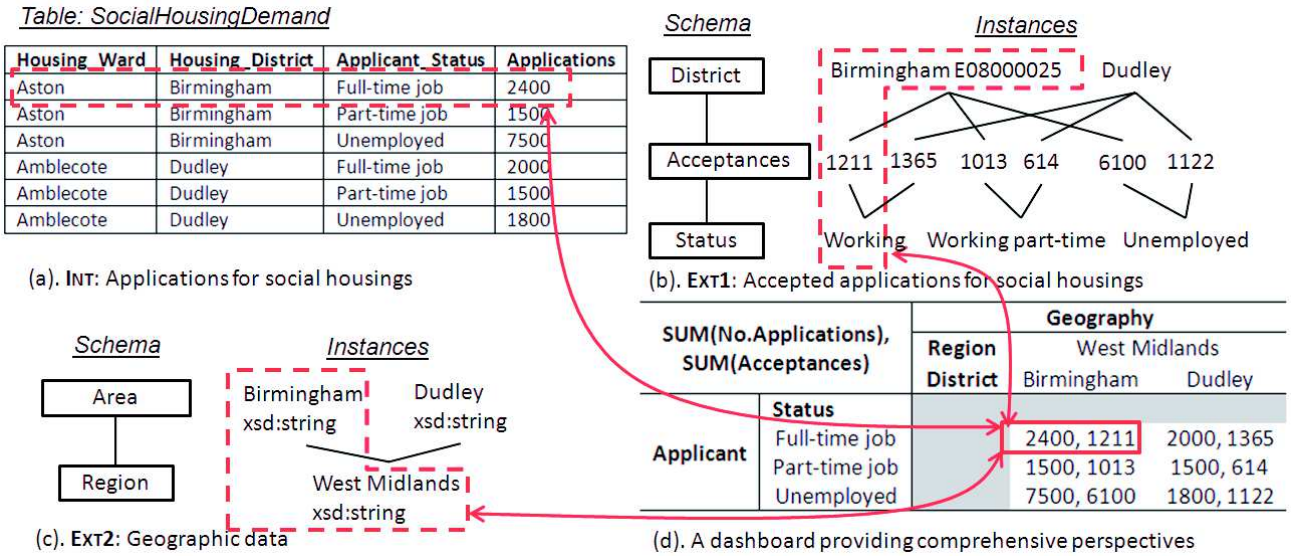


Figure 1: An extract of data in a DW and two LOD datasets

[1]http://linkeddata.org

[2]http://opendatacommunities.org/data/housingmarket/core/tenancies/econstatus

and *status* (cf. figure 1(b)). LOD1 follows a multidimensional structure expressed in *RDF Data Cube Vocabulary* (QB)[3] . The QB format only allows including one granularity in each analysis axis. The decision-maker needs new analysis possibilities to aggregate data based on multiple granularities. To discover more geographical granularities, the decision-maker looks into another dataset named LOD2. This dataset is managed by the *Office for National Statistics of the UK*[4]; it associates several *area*s (including districts) with one corresponding *region* (cf. figure 1(c)). Both LOD1 and LOD2 are real-world LOD which can be accessed through querying endpoints[5][6].

The above-mentioned warehoused data and LOD share some similar multidimensional features, as they are organized according to analysis subjects and analysis axes. However, analyzing data scattered in several sources is difficult without a unified data representation. During analyses, decision-makers must search for useful information in several sources. The efficiency of such analyses is low, since different sources may follow different schemas and contain different data instances. Facing these issues, the decision-maker needs a business-oriented view unifying data from both the DW and the LOD datasets. She/he makes the following requests regarding the view:

- An analysis subject should include all related numeric indicators from different sources, even though these indicators cannot be aggregated according to the same analytical granularities. To support real-time analyses, numeric indicators (e.g. *Applications* from the DW, *Acceptances* from the LOD1 dataset) and their descriptive attributes (e.g. *Housing_Ward*, *Housing_District* and *Applicant_Status* from the DW, *District* and *Status* from the LOD1 dataset) at different analytical granularities should be queried on-the-fly from sources;

- Analytical granularities related to the same analysis axis should be grouped together. For instance, the *Housing_Ward* and *Housing_District* granularities from the DW, the *District* granularity from the LOD1 dataset, the *Area* and *Region* granularities from the LOD2 dataset should be merged into one analysis axis;

- Attributes describing the same analytical granularity should be grouped together. The correlative relationships between instances of these attributes should be managed. For instance, the attribute *Housing_District* from the DW, the attribute *District* from the LOD1 dataset and the attribute *Area* from the LOD2 dataset should be all included in one analytical granularity related to *districts*. Correlative instances "*Birmingham*" from the DW, "*Birmingham E08000025*" from the LOD1 dataset and "*Birmingham xsd:string*" from the LOD2 dataset should be associated together, since they both refer to the same district;

- Summarizable analytical granularities should be indicated for each numeric indicator. For instance, only the measure *Applications* from the DW can be aggregated according the *Ward* analytical granularity. The other measure *Acceptances* from the LOD1 dataset is only summarizable starting from the *district* analytical granularity on the geographical analysis axis.

[3]http://www.w3.org/TR/vocab-data-cube
[4]https://www.ons.gov.uk/
[5]http://opendatacommunities.org/sparql
[6]http://statistics.data.gov.uk/sparql

Contribution. Our aim is to make full use of as much information as possible to support effective and well-informed decisions. To this end, we propose a unified view of data from both DWs and LOD datasets. At the schema level, the unified view should include in a single schema all information about an analysis subject described by all available analysis axes as well as all granularities (coming from multiple sources). At the instance level, the unified view should not materialize data that can be directly queried from the source. Nevertheless, it should manage the correlation relations between related attribute instances referring to the same real-world entity. With the help of the unified view, a decision-maker can easily obtain an overall perspective of an analysis subject. In the previous example, a unified view would enable decision-makers to analyze on-the-fly the number of *applications* and *acceptances* according to applicant's *status* and *district* as well as region (cf. figure 1(d)).

In this paper, we describe a generic modeling solution, named *Unified Cube*, which provides a business-oriented view unifying both warehoused data and LOD. Section 2 presents different approaches to unifying data from DWs and LOD datasets. Section 3 describes the conceptual modeling and graphical notation of *Unified Cube*s. Section 4 presents an implementation framework for *Unified Cube*s. Section 5 shows how analyses are carried out on a *Unified Cube* in a user-friendly manner. Section 6 illustrates the feasibility and the efficiency of our proposal through some experimental assessments.

2. Related work

Disparate data silos from different sources make decision-making difficult and tedious [43]. To provide decision-makers with an overall perspective during business analyses, an effective data integration strategy is needed. In accordance with our research context, we focus on work related to the integration of multidimensional data from DWs and LOD datasets. We classify existing researches into three categories.

The first category is named *ETL-based*. With the arrival of LOD, the BI community intuitively treated LOD as external data sources that should be integrated in a DW through ETL processes [15, 29, 36]. The obtained multidimensional DW is used as a centralized repository of LOD [38, 6]. Decision-makers can use classical DW analysis tools to analyze LOD stored in such DWs. However, the existing ETL techniques are inclined to populate a DW with LOD rather than updating existing LOD in a DW. No effective technique is proposed to guarantee the freshness of warehoused LOD presented to decision-makers during analyses. One promising avenue is to extend on-demand ETL processes [4] to fit for the integration of LOD in a DW at right time during business analyses. Otherwise, current ETL-based approaches are not suitable in today's highly dynamic context where large amounts of data are constantly published and updated; they collide with the distributed nature and the high volatility of LOD [24, 17].

The second category is named *semantic web modeling*. Since multidimensional models have been proven successful in supporting complex business analyses [35], the LOD community introduces new modeling vocabularies to semantically describe the multidimensionality of LOD through RDF triples. Among the proposed modeling vocabularies, the *RDF Data Cube Vocabulary*[7] (QB) is the current W3C

[7] http://www.w3.org/TR/vocab-data-cube

standard to publish multidimensional statistical data. The authors of [26] carry out multidimensional analyses over QB datasets. It is worth noticing that a QB dimension is a non-hierarchical concept. Due to this limit, the authors fail to support analyses involving multiple hierarchies within a dimension. To overcome this drawback, extensions of the QB vocabulary are needed to model a complete multidimensional schema. In [17], the authors propose the QB4OLAP vocabulary which adds more multidimensional characteristics to QB, like multiple analytical granularities within multiple aggregation paths and the specification of the aggregation functions associated with a measure. The authors of [37] present a multidimensional data model by blending together QB, SKOS[8] and RDFs [9] vocabularies. They also show how multidimensional analysis operations are translated into SPARQL queries based on the proposed data model. However, all semantic web modeling vocabularies are based on RDF formats which are primarily intended for machine consumption [37] and thus cannot be easily used by decision-makers. Even though the authors of [17, 37] provide conceptual models underlying semantic web modeling vocabularies, a user-oriented graphical notation is still missing to facilitate decision-makers' tasks of exploring RDF-like data schemas. Moreover, the work [26, 37] deals only with one LOD dataset. To handle data other than LOD, the work [17] uses mappings (i.e. R2RML) to populate a QB4OLAP dataset with warehoused data. Yet, no information is provided about if any of the above-mentioned work can integrate both warehoused data and LOD in one single dataset. Without solutions to these problems, current semantic web modeling approaches is not suitable for the unification of heterogeneous data in one user-oriented schema.

The third category is named *unified modeling*. It aims at providing generic data modeling solutions to (i) provide an overall representation of multisource data and (ii) manage relationships between multisource data. The authors of [1] envision a multidimensional model in which an internal database is gradually extended by fusing with external data, especially with data from the Web. The authors of [2] outline a new multidimensional model to support user-guided data discovery and acquisition of both internal warehoused data and external Web data. The unified modeling is a promising solution which allows blending data from multiple sources together. However, the work [1, 2] only describes general principles and research orientations. The authors of [28] propose IGOLAP vocabulary allowing representing multisource data according to a multidimensional structure. However, the compatibility of the work [28] is limited to LOD datasets. To deal with warehoused data, the authors suggest (i) transforming them into LOD formats through a mapping language proposed in [25] and (ii) loading large amounts of transformed data into a IGOLAP dataset. Consequently, the work [28] faces the same problems of the ETL-based work. Moreover, the feasibility of unifying multisource data in a IGOLAP dataset is only discussed without being experimentally assessed. It remains unknown how the complete unification process is instantiated and how heterogeneous instances from different sources are managed in a IGOLAP schema.

In this paper, we propose a generic multidimensional model which provides a unified view of both warehoused data and relevant multidimensional LOD. We denote our solution as *data unification* to differ from classical data integration solutions. Our data unification solution should break away from full integration of data into a stationary repository. It should keep a unified view over internal data (warehoused data) and external data (multidimensional LOD) without materializing all data. Based

[8] https://www.w3.org/TR/skos-reference/
[9] https://www.w3.org/TR/rdf-schema/

on the unified modeling, we propose a complete unification process including the implementation of a unified view and the processing of on-the-fly analyses on multisource data through a unified view.

3. Conceptual modeling of *Unified Cube*s

Unifying internal warehoused data and external LOD is not a straight forward task [33]. On one hand, DW and LOD communities do not share the same modeling paradigms. No existing data model allows blending multisource data without materializing them in a stationary repository. On the other hand, related data instances are scattered in different schemas. Existing modeling solutions without materialization do not allow managing the relations between heterogeneous instances from different sources. Facing these problems, we propose a new conceptual modeling solution, named *Unified Cube*, which is generic enough to unify (i) business data which are stored in multidimensional DWs from inside a company and (ii) multidimensional LOD which are stored in sources from outside a company. The *Unified Cube* modeling extends the classical multidimensional modeling to provide a single, comprehensive representation of multisource data. In the following section, we describe the *Unified Cube* model through an abstract representation. This representation is oriented towards designers who define a conceptual schema based on multisource data. A graphical notation of the abstract representation is proposed to facilitate the exploration of a *Unified Cube* by decision-makers.

3.1. Analysis subject: fact

Classically, a fact models an analysis subject. The fact is composed of a set of numeric indicators called measures. To support real-time analyses, the *Unified Cube* modeling extends the concept of measure by allowing on-the-fly extraction of measure values.

Definition 3.1. A fact is characterized by a name and a set of measures. It is denoted as $F = \{n^F; \mathsf{M}^F\}$ where:

- n^F is the fact's name;
- $\mathsf{M}^F = \{m_1; \ldots; m_p\}$ is a finite set of numeric indicators called *measure*s. Each measure m_e ($m_e \in \mathsf{M}^F$) is a pair $\langle n^{m_e}, E^{m_e} \rangle$, where n^{m_e} is the name of a measure and E^{m_e} is an *extraction formula* defined through a query algebra (e.g., relational algebra and SPARQL algebra[10]). The values of the measure m_e are denoted as *val*(m_e).

Remark. *Extraction formulae* enable on-the-fly querying of measures' values during analyses. The algebraic representation of extraction formula makes sure its compatibility with specific implementation environments of data source. Note that although the SPARQL algebra is not yet a W3C standard, it has already been integrated within several RDF querying framework. Each algebraic SPARQL expression is translated into one SPARQL query which is generic enough to work with all types of LOD datasets. Table 1 shows the algebraic form of commonly used SPARQL queries.

[10]https://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html

Table 1: SPARQL queries and their algebraic representation.

Query 1 SELECT * WHERE { ?s ?p ?o}	**Algebraic representation** (BGP (TRIPLE ?s ?p ?o))
Query 2 SELECT ?s ?p WHERE {?s ?p ?o}	**Algebraic representation** (PROJECT(?s ?p) (BGP (TRIPLE ?s ?p ?o)))
Query 3 SELECT ?o1 ?o2 WHERE{?s ?p ?o1. FILTER (?o1 < 5) OPTIONAL {?s ?p2 ?o2 . FILTER (?o2 > 10) }}	**Algebraic representation** (PROJECT(?o1 ?o2) (FILTER (< ?o1 5) (LEFTJOIN(BGP (TRIPLE ?s ?p ?o1)) (BGP (TRIPLE ?s ?p2 ?o2)) (> ?o2 10))))
Query 4 SELECT ?s (COUNT(?o) as ?nb) WHERE {?s ?p ?o} GROUP BY ?s HAVING (COUNT(?o) > 10)	**Algebraic representation** (PROJECT(?s ?nb) (FILTER (> ?.0 10) (EXTEND((?nb ?.0)) (GROUP (?s) ((?.0 (COUNT ?o))) (BGP (TRIPLE ?s ?p ?o))))))

Example. The fact named *Social Housings* contains two measures, namely $m_{Acceptances}$ and $m_{Applications}$. The measure $m_{Applications}$ has an extraction formula expressed in relational algebra, such as: $E^{m_{Applications}} = \mathtt{F}_{sum}(SocialHousingDemand.Applications)$. The extraction formula of the measure $m_{Acceptances}$ is defined upon SPARQL algebra, such as:

```
prefix m: <http://opendatacommunities.org/def/ontology/housing-market/core/>
prefix qb: <http://purl.org/linked-data/cube#>
prefix ex: <http://opendatacommunities.org/data/housing-market/core/tenancies/>

E^mAcceptances =
        (PROJECT (?Acceptances)
        |       (EXTEND ((?Acceptances?.0))
        |       |       (GROUP () ((?.0 (sum ?nbAcc)))
        |       |       |       (BGP    (TRIPLE?ob qb:dataSet ex: econstatus)
        |       |       |       |       (TRIPLE ?ob m:tenanciesObs ?nbAcc)
        |       |       |       )
        |       |       )
        |       )
        ).
```

3.2. Analysis axis: dimension

The concept of dimension in a *Unified Cube* follows the classical definition. A dimension may include a single or multiple analytical granularities. If several analytical granularities are defined, we can find one or several aggregation paths (also known as *hierarchies*).

Definition 3.2. A dimension corresponds to a one-dimensional space regrouping the analytical granularities related to one analysis axis. A dimension is denoted as D_i=$\{n^{D_i}; \mathbf{L}^{D_i}; \preccurlyeq^{D_i}\}$, where:

- n^{D_i} is the dimensions name;
- $\mathbf{L}^{D_i}$=$\{l_1; \ldots; l_k\}$ is a set of analytical granularities called levels;
- $\preccurlyeq^{D_i}$ is a *binary* relation which associates a child level l_a ($l_a \in \mathbf{L}^{D_i}$) with a parent level l_b ($l_b \in \mathbf{L}^{D_i}$), such as $l_a \preccurlyeq^{D_i} l_b$.

Example. We identify a dimension named *Geography* which groups all analytical granularities related to social housing's location. It includes three levels, such as $\mathbf{L}^{Geography} = \{l_{Geo.Ward}; l_{Geo.District}; l_{Geo.Region}\}$. The binary relation $\preccurlyeq^{Geography}$ reveals the aggregation paths (i.e., hierarchies) such as $l_{Geo.Ward} \preccurlyeq^{Geography} l_{Geo.District} \preccurlyeq^{Geography} l_{Geo.Region}$.

Our definition of dimension is generic enough to model a non-hierarchical dimension as well. A non-hierarchical dimension (e.g. D_{QB}) has only one level (e.g., $\mathbf{L}^{QB} = \{l_1\}$) including all the attributes of the dimension.

Two hierarchies from different sources do not always share a common lowest analytical granularity. Therefore, we remove the constraint of unique root level (i.e., $\exists_{=1} l_p \in \mathbf{L}^{D_i}, \forall l_q \in \mathbf{L}^{D_i} : l_p \preccurlyeq^{D_i} l_q$[11]) in the definition of a dimension. Without this constraint, a dimension may start at any level. This is an important property of a dimension regrouping levels from multiple sources, since measures from one source may only be analyzed according to a subset of levels coming from the same source. We define a *sub-dimension* as a part of dimension along which a measure can be summarized.

Definition 3.3. A *sub-dimension* of D_i, denoted $D_{i\backslash l_s} = \{ n^{D_{i\backslash l_s}}; \mathbf{L}^{D_{i\backslash l_s}}; \preccurlyeq^{D_i}\}$, corresponds to the part of the dimension D_i starting with the level l_s, where :

- $n^{D_{i\backslash l_s}}$ is the name of the sub-dimension;
- $\mathbf{L}^{D_{i\backslash l_s}}$ is the subset of levels, $\mathbf{L}^{D_{i\backslash l_s}} \subseteq \mathbf{L}^{D_i}, \forall l_i \in \mathbf{L}^{D_{i\backslash l_s}}, l_s \preccurlyeq^{D_i} l_i$;
- $\preccurlyeq^{D_i}$ is the same binary relation of the one on the dimension D_i.

Example. A sub-dimension of the geographical dimension is $D_{Geography\backslash l_{Geo.District}}$ named *Geography-District* with $\mathbf{L}^{D_{Geography\backslash l_{Geo.District}}}$=$\{l_{Geo.District}; l_{Geo.Region}\}$, which represents the subpart of the dimension $D_{Geography}$ that the measure $m_{Acceptances}$ from the LOD1 dataset is linked to.

[11] $\exists_{=1}$ represents the unique existential quantification meaning "there exists only one"

3.3. Analytical granularity: level

Classically, a level indicates a distinct analytical granularity described by a set of attributes from the same data source. In the context of *Unified Cube*s, the classical definition of level needs to be extended to group together attributes from different sources. Specifically, a level should manage a set of attributes by (i) indicating how attribute instances can be extracted from data sources and (ii) representing correlation relationships between related attribute instances from different sources.

Definition 3.4. A *level* represents an analytical granularity of a dimension. A level is denoted as $l_d = \{n^{l_d}; \mathsf{A}^{l_d}; C^{l_d}\}$, where:

- n^{l_d} is the levels name;
- $\mathsf{A}^{l_d} = \{a_1; \ldots; a_e\}$ is a finite set of *attribute*s. Each attribute a_x ($a_x \in \mathsf{A}^{l_d}$) is a pair $\langle n^{a_x}, E^{a_x} \rangle$, where n^{a_x} is the name of the attribute and E^{a_x} is an *extraction formula* indicating how instances of a_x can be extracted from the corresponding source. The domain of an attribute is denoted as $dom(a_x)$;
- $C^{l_d} : dom(a_x) \longrightarrow dom(a_y)(a_x \in \mathsf{A}^{l_d}, a_y \in \mathsf{A}^{l_d} \setminus a_x)$ is a symmetric transitive *correlative* mapping. It connects instances of the attribute a_x with equivalent instances of the attribute a_y, i.e. for an attribute instance $i_x \in dom(a_x)$, there exists at most one instance $i_y \in dom(a_y)$, such as $C^{l_d}(i_x) = i_y$.

Example. The level $l_{Economic.Status}$ on the dimension $D_{Applicant}$ contains a finite set of attributes $\mathsf{A}^{l_{Economic.Status}} = \{a_{Status}; a_{Applicant\_Status}\}$. To associate each attribute with its instances in data sources, two extraction formulae are defined within this level: $E^{a_{Applicant\_Status}} = \pi_{Applicant\_Status}(SocialHousingDemand)$ is linked to the attribute $a_{Applicant\_Status}$ from the DW, while the attribute a_{Status} has the following extraction formula:

```
prefix stat: <http://opendatacommunities.org/def/ontology/housing-market/core/tenancies/>
prefix qb: <http://purl.org/linked-data/cube#>
prefix ex: <http://opendatacommunities.org/data/housing-market/core/tenancies/>

E^aStatus =
        (PROJECT(?Status)
        |       (BGP (TRIPLE ?ob qb:dataSet ex: econstatus)
        |               (TRIPLE ?ob stat:econstatus  ?Status)
        |       )
        ).
```

The correlative mapping $C^{l_{Economic.Status}}$ associates the instances of the attribute a_{Status} with its equivalent instances of the attribute $a_{Applicant\_Status}$, such as:

$C^{l_{Economic.Status}}$: {{Working} $\longrightarrow$ {Full-time Job}; {Working part-time} $\longrightarrow$ {Part-time job}; {Unemployed} $\longrightarrow$ {Unemployed}}.

3.4. *Unified Cube*

In a *Unified Cube*, a fact and a dimension respectively include measures and levels from multiple sources. Each measure can be aggregated according to the set of levels from the same source. For each measure, decision-makers need to easily distinguish summarizable levels from non-summarizable levels. To do so, we propose the *level-measure* mapping which is flexible enough to associate each measure with a set of dimensions starting from any level.

Definition 3.5. A *Unified Cube* is a n-dimensional finite space describing a fact with some dimensions. It is denoted as $UC = \{F; \mathbf{D}; \mathsf{LM}\}$, where:

- F is a *fact* containing a set of *measures*;
- $\mathbf{D}=\{D_1; \ldots; D_n\}$ is a finite set of *dimensions*;
- $\forall m_e \in \mathsf{M}^F$, LM: $2^{\mathbf{L}^1{}_{\setminus l_p} \times \ldots \mathbf{L}^n{}_{\setminus l_q}} \longrightarrow m_e$ is a *level-measure* mapping which associates a subset of summarizable analytical granularities (i.e., levels) with a measure m_e, such as $\forall i \in [1..n], \mathbf{L}^i{}_{\setminus l_s} (l_s \in \mathbf{L}^i)$ corresponds to a subset of levels on the dimension $D_i (D_i \in \mathbf{D})$ which starts from the level l_s.

We propose a graphical notation for *Unified Cube*s by extending the *star schema* notation proposed in [21]. The modifications are as follows:

- A fact is represented by a rectangle. The fact name is situated within the rectangle on top;
- A measure is circled by a rectangle;
- A dimension is enclosed by a rectangle. The name is placed above a dimension;
- Each level is represented by a circle with all its descriptive attributes lying below;
- A binary relation is represented by an arrow from a child level to a parent level;
- A level-measure mapping is represented by a line between a measure and a dimension starting from any level. In order to simplify the notation, the level-measure mapping is drawn between a measure and its lowest summarizable levels within corresponding dimensions.

Example. The *Unified Cube* of our case study contains two dimensions $\mathbf{D} = \{D_{Applicant}, D_{Geography}\}$. The two measures of the fact named *Social Housings* are associated with their summarizable levels, i.e., LM: $\{ \{\mathbf{L}^{Applicant}; \mathbf{L}^{Geography}{}_{\setminus l_{Geo.District}}\} \longrightarrow \{m_{Acceptances}\}; \{\mathbf{L}^{Applicant}; \mathbf{L}^{Geography}\} \longrightarrow \{m_{Applications}\} \}$. The complete graphical notation of this *Unified Cube* is shown in figure 3.

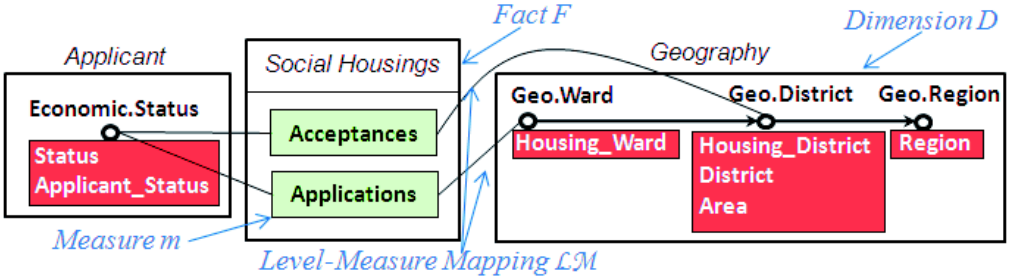


Figure 3: Graphical notation of a *Unified Cube*

3.5. Main contributions of *Unified Cube* modeling

To the best of our knowledge, *Unified Cube* is the first model that allows unifying appropriate data for decision-making from both DWs and LOD datasets without materializing all data in a stationary repository. By including business-oriented concepts and a graphical notation, a *Unified Cube* can support analyses on multiple data sources in a user-friendly way without requiring specialized knowledge on logical or physical data modeling. The *Unified Cube* modeling breaks through three obstacles in the multidimensional modeling field: (i) warehoused data and LOD can be queried on-the-fly during analyses through instance-finding mappings (i.e. extraction formulae of measures and attributes), (ii) a measure can be linked to a dimension starting from any level through intra-aggregate mappings (i.e. level-measure mapping LM), and (iii) attribute instances from one source are linked with equivalent instances of attribute from another source through inter-instance mappings (i.e. correlative mappings C^{l_d}). The powerful *Unified Cube* modeling is further coupled with a user-friendly graphical notation, so that non-expert users can easily explore by themselves the overall schema of multisource data during analyses.

4. Implementation of *Unified Cube*s

A conceptual *Unified Cube* provides a generic representation unifying data which are physically stored in different sources. Once defined by a schema designer, a *Unified Cube* needs to be implemented before being used to support analyses on multisource data. To do so, we provide a framework which automates the implementation of *Unified Cube*s. Two modules are identified within the framework, namely *schema* and *instance*. The *schema* module aims at managing the overall structure of a *Unified Cube* (cf. section 4.1), while the *instance* module serves as an integrated repository of correlative attribute instances from heterogeneous sources (cf. section 4.2).

4.1. Schema module

The *schema* module manages the multidimensional representation of a *Unified Cube*. It serves as an interface between business-oriented concepts in a *Unified Cube* and referents in data sources. To do so, we propose a metamodel which offers a uniform way to access different data sources through concepts in a *Unified Cube*. In this section, we firstly describe the components of the metamodel. Then, we propose an algorithm to automatically instantiate the metamodel.

In the metamodel, concepts such as fact, measure, dimension, level and attribute are modeled through classes. Composition relationships are used to associate a containing class (e.g., fact) with a contained class (e.g., measures). Binary relations between parent and child levels are represented by a recursive association connected to the *Level* class. Level-measure mappings are managed by the association between classes named *Measure* and *Level*. It is worth noticing that extraction formulae of measures and attributes are translated into executable queries (i.e., queryM and queryA). A querying endpoint is associated with each measure and each attribute, so that data instances can be extracted on-the-fly during analyses. The class diagram of the metamodel is shown in figure 4.

We complete the proposed metamodel with an instantiation process (cf. algorithm 1). The goal is to automatically instantiate classes and associations of the metamodel based on a conceptual *Unified*

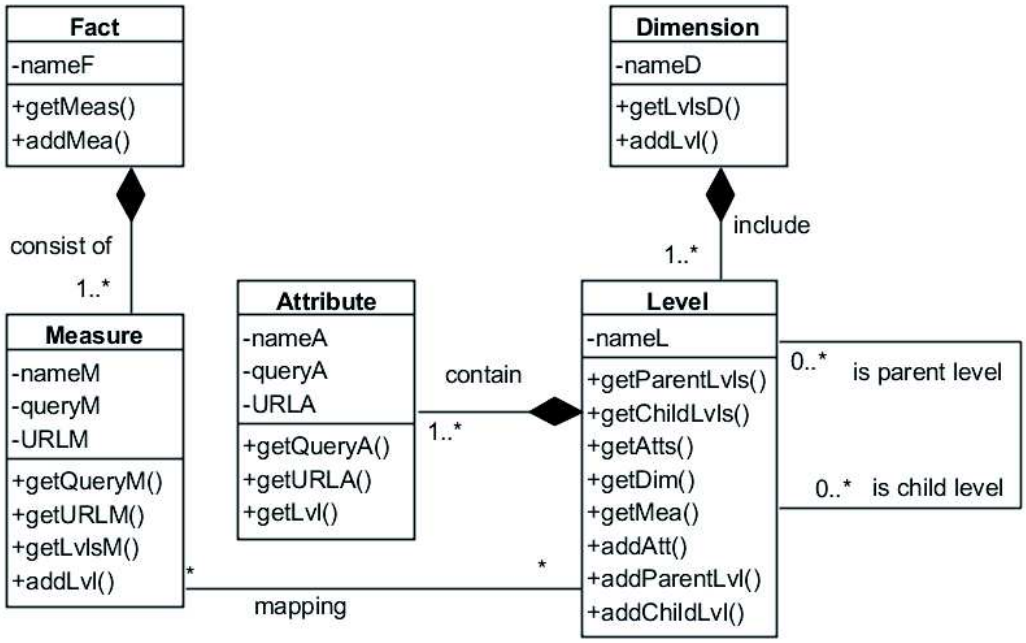


Figure 4: Class diagram of the metamodel for *Unified Cube*s

Cube. First, the algorithm instantiates the dimension class and associates each dimension instance (D_i^{meta}) with related attributes instances (a_x^{meta}) organized according to child (l_e^{meta}) and parent (l_f^{meta}) level instances (cf. lines 1 - 16). Then, the algorithm creates a fact instance (F^{meta}) and links it with a set of measure instances (m_g^{meta}) (cf. lines 17 - 22). At last, the algorithm instantiates associations between a measure instance (m_g^{meta}) and its summarizable level instances ($\boldsymbol{L}^r_{\setminus l_h} \times \ldots \times \boldsymbol{L}^t_{\setminus l_k}$) (cf. lines 23 - 26). The output of the instantiation process is an instantiated metamodel which manages the overall structure of a *Unified Cube*.

Specifically, each *Unified Cube* dimension is used to instantiate a *Dimension* class (cf. lines 1 and 2). For each level on the *Unified Cube* dimension, a *Level* class is instantiated and associated with the corresponding *Dimension* class (cf. lines 3 - 5). The extraction formula of an attribute in the *Unified Cube* is transformed into an equivalent algebraic operation before being translated into an executable query. An *Attribute* class is instantiated with a name, a query and a query endpoint. It is then associated with the corresponding level (cf. lines 6 - 11). Binary relations within a dimension are used to instantiate the association between a child level instance and a parent level instance (cf. lines 13 - 15). A *Fact* class is instantiated (cf. lines 17). An executable query is generated for each measure based on the extraction formula. With the measure's name, translated query as well as query endpoint, a *Measure* class is instantiated and associated with the fact (cf. lines 18 - 22). The *level-measure* mappings in the *Unified Cube* are used to instantiate the association between a measure instance and a set of level instances (cf. lines 23 - 26). Note that the operations used in the algorithm can be found in the metamodel in figure 4.

Example. We apply the algorithm to the *Unified Cube* of our case study (cf. figure 3). The instantiated metamodel contains (i) one instance of *Fact* (ii) two instances of *Measure*, (iii) two instances of

Algorithm 1: Metamodel Instantiation

input : A *Unified Cube* = {F; $\boldsymbol{D}$; $\boldsymbol{LM}$}
output: An instantiated metamodel

for *each* $D_i \in \boldsymbol{D}$ **do**
  Instantiate the Dimension class: $D_i^{meta} = newDimension(n^{D_i})$;
  for *each* $l_d \in \boldsymbol{L}^{D_i}$ **do**
    Instantiate the Level class: $l_d^{meta} = newLevel(n^{l_d})$;
    $D_i^{meta}.addLvl(l_d^{meta})$;
    for *each* $a_x \in \boldsymbol{A}^{l_d}$ **do**
      Translate the extraction formula E^{a_x} *into a query* Q^{a_x};
      Get the attribute's query endpoint URL^{a_x};
      Instantiate the Attribute class: $a_x^{meta} = newAttribute(n^{a_x}; Q^{a_x}; URL^{a_x})$;
      $a_x^{meta}.addLvl(l_d^{meta})$;
    end
  end
  for *each* $l_e \preccurlyeq^{D_i} l_f$ **do**
    $l_e^{meta}.addParentLvl(l_f^{meta}), l_f^{meta}.addChildLvl(l_e^{meta})$;
  end
end
Instantiate the Fact class: $F^{meta} = newFact(n^F)$;
for *each* $m_g \in \boldsymbol{M}^F$ **do**
  Translate the extraction formula E^{m_g} *into a query* Q^{m_g};
  Get the measure's query endpoint URL^{m_g};
  Instantiate the Measure class: $m_g^{meta} = newMeasure(n^{m_g}; Q^{m_g}; URL^{m_g})$;
  $F^{meta}.addMea(m_g^{meta})$;
  Find levels $\boldsymbol{L}^r_{\setminus l_h} \times \ldots \times \boldsymbol{L}^t_{\setminus l_k}$ *associated with* m_g *through level-measure mappings* LM*, such as* $\boldsymbol{L}^r_{\setminus l_h} \times \ldots \times \boldsymbol{L}^t_{\setminus l_k} \subseteq \boldsymbol{L}^1 \times \ldots \times \boldsymbol{L}^n$*,* LM*:* $\boldsymbol{L}^r_{\setminus l_h} \times \ldots \times \boldsymbol{L}^t_{\setminus l_k} \longrightarrow m_g$;
  for *each level* $l_h \in \boldsymbol{L}^r_{\setminus l_h} \times \ldots \times \boldsymbol{L}^t_{\setminus l_k}$ **do**
    $m_g^{meta}.addLvl(l_h^{meta})$;
  end
end

Dimension, (iv) four instances of *Level* with two associations implementing binary relations and seven associations implementing level-measure mappings and (v) seven instances of *Attribute*.

In figure 5, a snapshot of the instantiated metamodel is presented in the form of an object diagram. For the sake of readability, the snapshot only includes some components of the *Unified Cube* of the case study. In the snapshot, the *Geography* dimension includes the level *Geo.District*. This level contains three attributes from the DW and the two LOD datasets, namely *District*, *Housing_District* and *Area*. The measure named Acceptances of the fact named *Social_Housings* is mapped to the summarizable level *Geo.District* through level-measure mappings.

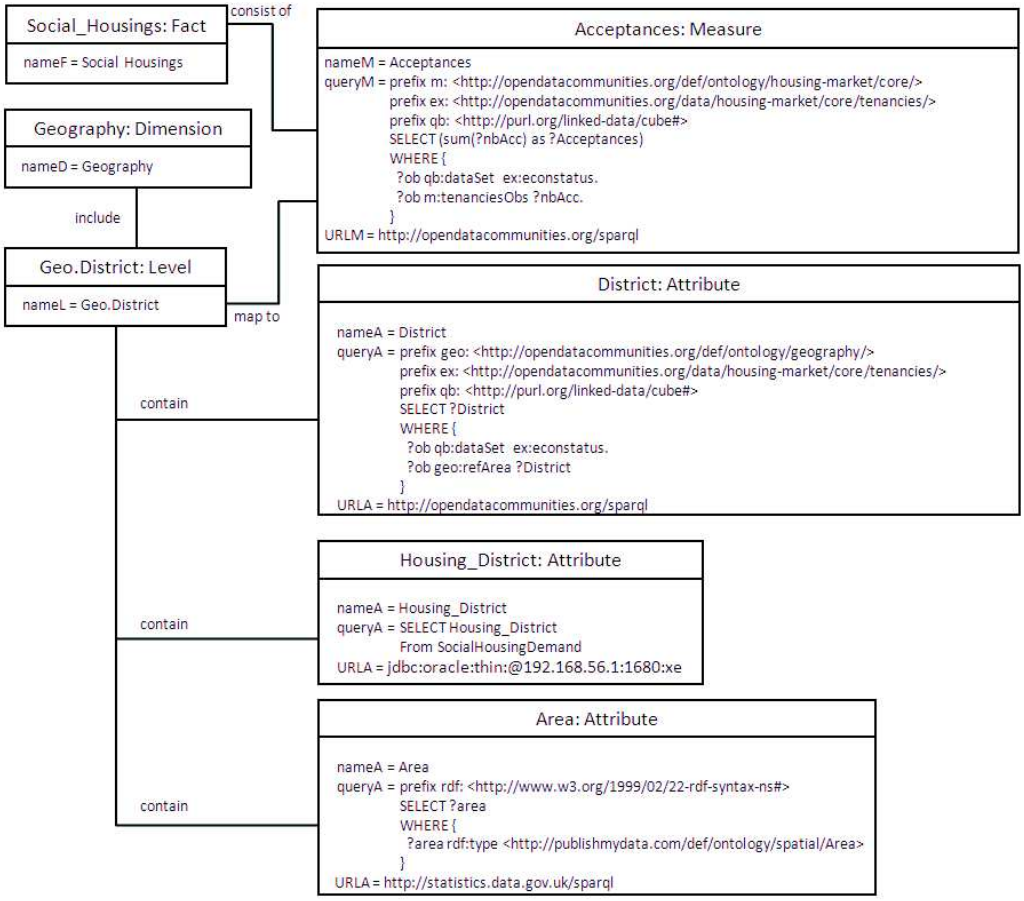


Figure 5: Snapshot of instantiated metamodel

4.2. Instance module

In a *Unified Cube*, equivalent attribute instances from different sources are associated together by correlative mappings (cf. section 3.3). Due to different understandings of the same real-world concepts, equivalent attribute instances may take heterogeneous forms in different data sources. It is necessary to bridge the differences among multiple data sources at the instance level. To do so, we propose a table of correspondences which manages correlative mappings between related attribute instances.

Definition 4.1. A *table of correspondences* links up pairs of correlative attribute instances by annotating each pair with a confidence score. Its schema is denoted as $T = \{id; n^{a_x}; i^{a_x}; n^{a_y}; i^{a_y}; score^{xy}\}$, where:

- id is an *identifier*;
- n^{a_x} is the name of the attribute a_x;
- i^{a_x} is an instance of the attribute a_x, $i^{a_x} \in dom(a_x)$;
- n^{a_y} is the name of the attribute a_y situated within the same level as the attribute a_x;

- i^{a_y} is an instance of the attribute a_y, $i^{a_y} \in dom(a_y)$;
- $score^{xy}$ is a normalized confidence score between i^{a_x} and i^{a_y}, $score^{xy} \in [0, 1]$.

Populating a table of correspondences requires identifying correlative instances from different sources. In some cases, correlative mappings are already semantically represented between sources. For instance, it is common to find in a LOD dataset that the OWL property `owl:sameAs` is used to associate an instance with an equivalent instance in another LOD dataset. A table of correspondences is populated by directly following such existing correlative mappings.

In other cases, there exists no correlative mapping between two sources (e.g. between a DW and a LOD dataset). We rely on techniques of instance matching to identify correlative instances between two datasets. Instance matching belongs to the field of entity resolution [12, 5]. Existing entity resolution processes are designed for matching instances from sources of the same type. Classically, problems of matching instances from sources of different types are handled in a simplistic way by transforming heterogeneous sources into a common format and then following a matching process designed for homogeneous sources. However, as more and more data are involved in matching nowadays, it is not efficient to carry out matching with a transformation stage. The matching should be performed directly between different sources, regardless if the sources belong to the same type or not.

To do so, we propose a process based on techniques which are generic enough to enable direct matching of instances from heterogeneous sources. As shown in figure 6, the process starts with a stage which generates matching candidates directly from raw data. The second stage consists of calculating the similarity between matching candidates. The final stage determines the best matches between the two sets of instances.

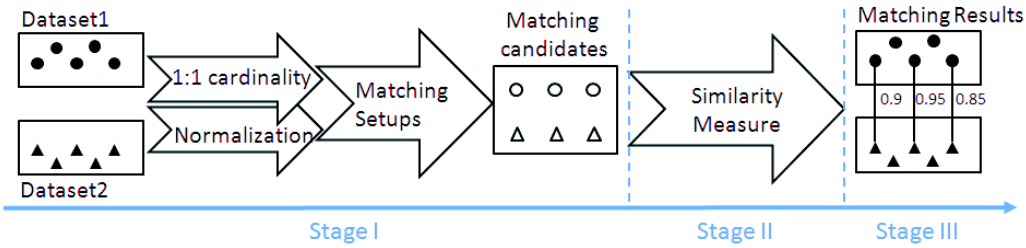


Figure 6: Process of matching between two sets of instances from different sources

Specifically, during the first stage, we apply some processing techniques to prepare attribute instances for the matching. First, we fix the matching cardinality to 1:1 (i.e. injective mapping), which assumes one instance in a source can be matched with only one instance in the other source. Second, we normalize descriptive information of instances by eliminating stylistic differences due to capitalization, punctuation, and non-Latin characters. Third, we propose several solutions to set up the matching candidates in string form.

- A straightforward setup consists of concatenating all descriptive information into one long string. We denote this setup as *Concatenated*. The opposite setup is to compare individually descriptive information of the same kind [27]. We denote this setup as *Separated*.

- Some descriptive information may contain useless parts, e.g., name spaces such as the prefix eg:[12]. We propose a setup named *Optimized* which removes useless parts to keep only information related to matching [10]. The opposite setup is denoted *Unprocessed*, as it takes descriptive information "As-Is" without any modification.

Remark. In the case where each instance is described by one unique string, the matching is carried out directly with the unique description of an instance without concatenation.

During the second stage, we rely on string similarity measures to calculate a confidence score between 0 and 1 for each pair of matching candidates. String similarity measures are widely used as syntax-based matching technique in the field of entity resolution [19], for instance, record linkage between warehoused data [7] and instance matching between LOD [9]. We identify sixteen widely-used string similarity measures in the scientific literature. We carry out some experimental assessments to find out the most efficient string similarity measures. Based on the results, we identify three string similarity measures, namely *N Grams Distance* [40], *Levenshtein* [44], and *Smith Waterman* [30], which are generic and efficient enough to fit for various matching tasks. Details about our experimental assessments will be presented in section 6.

Remark. Matching based on string similarity measures is often reinforced by some auxiliary techniques. One of the most widely used auxiliary techniques is semantic-based. It relies on formally described semantics to perform deductive matching methods. However, in nowadays open, evolving world, different sources usually adopt different methods to describe the semantics. Matching two sources with well-defined semantics through semantic-based techniques is already hard enough, e.g. ontology and instance matching tracks of *Ontology Alignment Evaluation Initiative*[13], not to mention difficulties in matching data from semantic-light sources, such as DWs. A DW's semantics are not rich enough, since they are defined during the design phase and not explicitly represented after implementation [18]. An intermediate ontology is generally used to provide additional semantics of warehoused data [20]. Building an intermediate ontology usually requires the intervention of domain experts and thus is hardly automated. Moreover, the slightest error in an intermediate ontology can introduce hidden bias in matching. Therefore, semantic-based techniques using an intermediate ontology are neither generic nor efficient enough to be applied to instance matching in a *Unified Cube*.

During the third stage, we determine the best matches by finding 1:1 matching between two sets of instances. We apply the *stable-marriage* algorithm [22] to identify mutually accepted matching between two datasets. The algorithm repeats the following steps until all instances in two datasets are matched: let D_1 and D_2 be two datasets, (i) an unmatched instance in D_1 establishes a mapping with the most similar instance in D_2 if no previous mapping is broken between them and (ii) an instance in D_2 accepts a mapping from D_1 and breaks an existing mapping (if any) when the new mapping comes from a more similar instance in D_1. A set of best matches between two data sources is produced. Then, the best matches are used to populate the table of correspondences.

Example. In the *Unified Cube* of our case study, the level $l_{Geo.District}$ includes three attributes from different sources. Correlative instances are identified in the following ways:

[12] eg:¡http://www.example.org/¿

[13] http://oaei.ontologymatching.org/

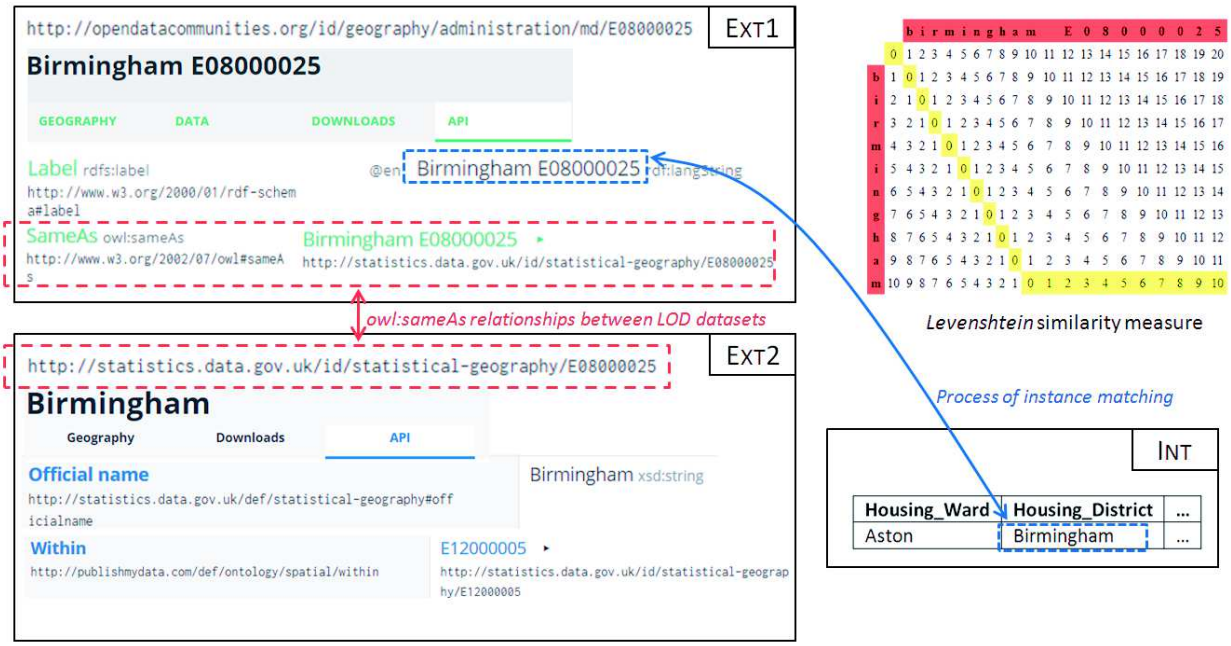


Figure 7: Identifying equivalent attribute instances within the level $l_{Geo.District}$

- the attributes named *District* and *Area* come from the LOD1 and LOD2 datasets respectively. In the LOD1 dataset, each district is linked to an equivalent district in the LOD2 dataset through the OWL property owl:sameAs (cf. figure 7). By referring to the links between LOD datasets, 243 pairs of districts from the LOD1 dataset and the LOD2 dataset are associated together with a perfect confidence score (i.e. score = 1);
- *Housing_District*'s instances from the DW share similar description with instances of *Area* from the LOD1 dataset. Yet, there is no existing correlative mapping between the DW and the LOD1 dataset. We apply our proposed instance matching process to identify correlative instances of the attributes *Housing_District* and *Area*.

A snapshot of the obtained table of correspondences is shown in figure 8.

ID	ATTRIBUTE	INSTANCE	EQUIVATTRIBUTE	EQUIVINSTANCE	SCORE
1	District	BirminghamE08000025	Area	Birminghamxsd:string	1
2	District	BirminghamE08000025	Housing_District	Birmingham	0.476

Figure 8: A snapshot of the table of correspondences

The advantage of our implementation framework is twofold. On one hand, instances are not materialized and can be queried on-the-fly. In this way, the data freshness is guaranteed during analyses. On the other hand, data from different sources can be analyzed in a unified way owing to the table of correspondences. The cost of maintaining a table of correspondences is quite low, as attribute instances only represent 1% to 6% of a multidimensional dataset's size [42].

5. Analysis processing of *Unified Cube*s

Our proposed implementation framework enables interactions between a *Unified Cube* and multiple data sources at both schema and instance levels. Based on the framework, we propose an analysis processing process enabling decision-making on multiple sources in a user-friendly way.

Processing analyses on multiple sources has been studied by classical mediator approaches. As shown in figure 9(a), in the mediator-based approach, a query posed over a global schema (i.e. Q_{global}) is transformed into local queries (i.e. Q_{local}) by a mediator. A wrapper translates a local query into an executable query in one source. Query results are transformed into a common form (e.g., relational data) to generate partial results of analysis. All partial results are then combined together to form a final analysis result [3].

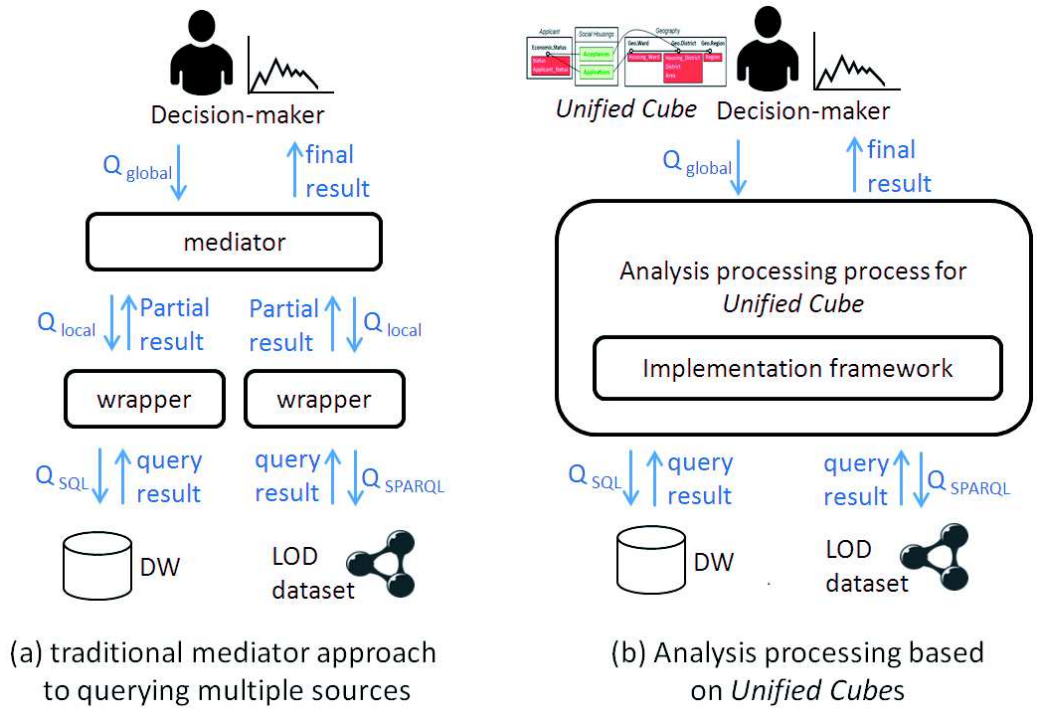


Figure 9: Mediator-based approach versus analysis processing process for *Unified Cube*

Comparing to classical mediator-based approaches, our proposed analysis processing process is saved from the intermediate steps in wrappers: executable queries are directly generated based on a global query posed over a *Unified Cube*, while a final analysis result is directly built from extracted multisource data. Specifically, at the beginning of an analysis, a decision-maker explores a *Unified Cube* and expresses a global query by choosing a set of attributes and measures related to the analysis. The analysis processing process automatically generates executable queries in multiple sources by using the extraction formulae stored in the implementation framework. After the execution of queries, data from several query results are automatically blended together by referring to the correlative relations provided by the implementation framework. At the end of an analysis, the decision-maker receives one unique analysis result including data from multiple sources.

In this section, we first describe how queries are automatically generated for an analysis (cf. section 5.1). Then, we present how one analysis result is generated based on data extracted from different sources (cf. section 5.2). At last, we illustrate the feasibility of our proposal through a prototype analysis framework (cf. section 5.3).

5.1. Queries generation

With the help of a *Unified Cube*, a decision-maker can express an analysis need by choosing a set of measures and attributes. To facilitate decision-makers tasks, we propose a process whose goal is to extract data related to an analysis from multiple sources (cf. algorithm 2). This is done through the generation of queries in each data source involved in an analysis.

First, among the chosen attributes, the algorithm finds a subset of attributes ($\boldsymbol{A^M}$) that are directly related to chosen measures ($\boldsymbol{M}$) (cf. lines 1 - 23). Then, for each chosen measure (m_e), the algorithm generates a query by combining its extraction formula and those of the related attributes ($\boldsymbol{A}^{m_e}$) (cf. lines 24 - 30). At last, if there are some chosen attributes ($\boldsymbol{A} \setminus \boldsymbol{A^M}$) that are not directly related to any chosen measure, their extraction formulae are added in the set of generated queries (cf. lines 31 - 35). The output of the algorithm is a set of queries extracting data related to an analysis from multiple sources. The abstract notation of *Unified Cube*s and operations involved in the algorithm are described in previous sections 3 and 4.1 respectively.

Specifically, in a *Unified Cube*, each measure is summarizable with regard to a set of levels. The set of summarizable levels may differ from one measure to another, while not every measure can be calculated according to every level (e.g. figure 3). If an analysis mistakenly involves a measure and attributes within non-summarizable levels, the algorithm displays a warning message and breaks the execution (cf. lines 7 - 8). Otherwise, the algorithm classifies measures and attributes by data source. Specifically, for each measure m_e, the algorithm finds a list of attributes from the same sources (i.e. $\boldsymbol{A}^{m_e}$ in line 4). Such attributes may (i) be an attribute ax chosen by decision-makers (cf. lines 10 and 11), (ii) correspond to an attribute within the same level of a_x (i.e. a_y at lines 14 and 15) and (iii) come from the set of attributes within the highest level linked to m_e (i.e. a_{y_bis} at lines 17 and 18). Extraction formulae of a measure and its attributes are then grouped together (cf. lines 24 - 26). As some analyses involve multiple levels, grouping predicates have to be added in generated queries (cf. line 27). During the execution, the algorithm picks out attributes linked to a chosen measure (i.e. $\boldsymbol{A^M}$ in lines 1 and 23). For the other attributes (i.e. $\boldsymbol{A} \setminus \boldsymbol{A^M}$ in line 32), the algorithm directly adds their extraction formulae in the set of queries (i.e. $\boldsymbol{Q}$). Each query in the output result includes measures and attributes from a single source.

To better explain how the algorithm works, we show some examples of analysis based on the *Unified Cube* of our case study (cf. figure 3).

The first analysis includes a measure and an attribute from different sources. The algorithm firstly finds the set of levels linked to the measure $m_{Acceptences}$ (cf. line 3, $\boldsymbol{L}^{m_{Acceptences}} = \boldsymbol{L}^{Geography}{}_{\setminus l_{Geo.District}} \cup \boldsymbol{L}^{Applicant}$. Then it retrieves the chosen attribute's level (cf. lines 6, $l^{a_{Housing\_Ward}}$). By comparing $l^{a_{Housing\_Ward}}$ with $\boldsymbol{L}^{m_{Acceptences}}$, the algorithm finds that the chosen measure is not linked to the level of the chosen attributes (cf. line 7, $l^{a_{Housing\_Ward}} \notin \boldsymbol{L}^{m_{Acceptences}}$). It displays a warning message and stops executing (cf. line 8). No query is generated for the first

Algorithm 2: Automatic Query Generation

input : A set of measures $\boldsymbol{M}$; a set of attributes $\boldsymbol{A}$
output: A set of generated queries $\boldsymbol{Q}$ = $\{q_1; \ldots q_n\}$

Create an empty set of attributes $\boldsymbol{A}^{\boldsymbol{M}}, \boldsymbol{A}^{\boldsymbol{M}} = \varnothing$;
for *each* $m_e \in \boldsymbol{M}$ **do**
  Get all levels linked to m_e: $\boldsymbol{L}^{m_e} = m_e.getLvlsM()$;
  Create an empty list of attributes $\boldsymbol{A}^{m_e}, \boldsymbol{A}^{m_e} = \varnothing$;
  for *each* $a_x \in \boldsymbol{A}$ **do**
    Get the level of a_x: $l^{a_x} = a_x.getLvl()$;
    if $l^{a_x} \notin \boldsymbol{L}^{m_e}$ **then**
      Impossible analysis, break the execution and display a warning message;
    else
      if $a_x.getURLA() = m_x.getURLM()$ **then**
        Add the attribute a_x *in the list* $\boldsymbol{A}^{m_e}$;
      else
        Get the set of attributes on the level l^{a_x}: $\boldsymbol{A}^{l^{a_x}} = l^{a_x}.getAtts()$;
        if $\exists a_y \in \boldsymbol{A}^{l^{a_x}}$ *such as* $a_y.getURLA() = m_e.getURLM()$ **then**
          Add the attribute a_y *in the list* $\boldsymbol{A}^{m_e}$;
        else
          Find an attribute a_{yBis}, such as $a_{yBis}.getLvl() \in l^{a_x}.getChildLvls() \wedge \nexists a_z \in a_{yBis}.getLvl().getParentLvl().getAtts(), a_z.getURLA() = m_e.getURLM()$;
          Add the attribute a_{yBis} *in the list* $\boldsymbol{A}^{m_e}$;
        end
      end
    end
  end
  $\boldsymbol{A}^{\boldsymbol{M}} \longleftarrow \boldsymbol{A}^{\boldsymbol{M}} \bigcup \boldsymbol{A}^{m_e}$;
  Get the query of the measure m_e: $q^{m_e} = m_e.getQueryM()$;
  for *each* $a_p \in \boldsymbol{A}^{m_e}$ **do**
    Add $a_p.getQueryA()$ *in* q^{m_e};
    Add a_p *in the grouping predicates in* q^{m_e} ;
  end
  Add q^{m_e} *in* $\boldsymbol{Q}$;
  if $\boldsymbol{A} \setminus \boldsymbol{A}^{\boldsymbol{M}} \neq \varnothing$ **then**
    for *each* $a_x \in \boldsymbol{A} \setminus \boldsymbol{A}^{\boldsymbol{M}}$ **do**
      Add $a_k.getQueryA()$ *in* $\boldsymbol{Q}$
    end
  end
end

Table 2: Analysis needs with corresponding measures and attributes

Analysis need	**DW**	**LOD1**	**LOD2**	**Comment**
1. number of accepted applications by housing ward.	$a_{Housing\_Ward}$	$m_{Acceptences}$	n/a	impossible
2. number of accepted applications by economic status.	n/a	$m_{Acceptences}$, a_{Status}	n/a	direct querying
3. number of accepted applications by applicant's status.	$a_{Applicant\_Status}$	$m_{Acceptences}$	n/a	multisource
4. number of accepted applications by applicant's region.	n/a	$m_{Acceptences}$	a_{Region}	multisource, adaptation
5. number of submitted and accepted applications by applicant's status and housing district, region.	$m_{Applications}$ $a_{Applicant\_Status}$ $a_{Housing\_District}$	$m_{Acceptences}$	a_{Region}	direct querying, multisource, adaptation

analysis. Through the first example, we can see our proposed algorithm is reliable enough to detect analysis needs that are not supported.

The second analysis corresponds to a classical involving only one data source. In this case, measures and attributes can be directly queried from the source. Specifically, after verifying that $m_{Acceptences}$ and a_{Status} come from the same source, the algorithm combines their extraction formulae together (cf. lines 10 and 11). The generated query in appendix A shows a *Unified Cube* also supports analyses involving one single data source.

The third analysis calculates a measure according to an attribute from a different source. The algorithm first retrieves all attributes within the same level of $a_{Applicant\_Status}$ (cf. line 13, $\boldsymbol{A}^{l^{a_{Applicant\_Statue}}}$) = $\{a_{Applicant\_Status}, a_{Status}\}$). An attribute a_{Status} is found in the same source of measure $m_{Acceptences}$ (cf. line 14, $a_{Status} \in \boldsymbol{A}^{l^{a_{Applicant\_Statue}}}$, a_{Status}.getURLA()= $m_{Acceptences}$.getURLM()). The extraction formulae of $m_{Acceptences}$ and a_{Status} are combined together in the same way of the second analysis (cf. line 15). The algorithm finds the relative complement of $\boldsymbol{A}^{l^{a_{Applicant\_Statue}}}$ in chosen attribute set A is not empty (cf. line 31, $\boldsymbol{A} \backslash \boldsymbol{A}^{l^{a_{Applicant\_Statue}}}$=$\{a_{Status}\}$). The extraction formula of a_{Status} is added in the set of generated queries (cf. line 33). Appendix B shows the two generated queries of the third analysis.

The fourth analysis involves two different sources. No attribute within the level of a_{Region} belongs to the source of $m_{Acceptances}$ (cf. lines 13 and 14, $\nexists a_y \in l^{a_{Region}}$.getAtts(), a_y.getURLA() $\neq$ $m_{Acceptances}$.getULLM()). Adaptations are needed to generate executable queries for each source. To do so, the algorithm finds $a_{District}$ which belongs to a lower level of $l^{a_{Region}}$ and the highest level available in the source of $m_{Acceptances}$ (cf. line 17, $l^{a_{District}} \in l^{a_{Region}}$.getChildLvls()

$\wedge \forall a_z \in l^{a_{District}}$.getParentLvl().getAtts(), a_z.getURLA() $\neq m_{Acceptances}$.getURLM()). The first query is generated by combining together the extraction formulae of $a_{District}$ and $m_{Acceptances}$ (cf. line 18), while the second query is generated by directly using the extraction formula of a_{Region}. Both queries are present in appendix C.

The fifth analysis unifies measures and attributes from three different sources. The extraction formulae of $m_{Applications}$, $a_{Applicant\_Status}$ and $a_{Housing_District}$ are directly combined together to generate the first query, since they come from the same DW (cf. lines 10 and 11). Then, the algorithm finds attributes a_{Status} and $a_{District}$ from the LOD1 dataset (cf. lines 13, 14 and 17). The second query is generated by referring to the extraction formulae of $m_{Acceptances}$, a_{Status} and $a_{District}$ (cf. lines 15 and 18). The third query refers directly to the extraction formula of a_{Region} (cf. lines 31 - 34). The last analysis covers all previously discussed scenarios: from a single source to multiple sources with and without adaptation. Three generated queries are shown in appendix D.

5.2. Analysis result generation

After the execution of generated queries, several query results are returned from different sources. It is difficult for decision-makers to obtain an overall perspective from different query results containing data of different types. It is necessary to provide one unified result including all data related to the analysis. To do so, we propose a generic modeling solution of query result and analysis result which can be considered as a set of attribute instances possibly linked to a set of measure values.

Definition 5.1. A result is denoted as $R_i = \{\boldsymbol{V}^{R_i}; \boldsymbol{I}^{R_i}; f^{R_i}\}$, where:

- $\boldsymbol{V}^{R_i} = \varnothing \vee \{val(m_1), \ldots, val(m_f)\}$ is an empty set or a set of measure values;
- $\boldsymbol{I}^{R_i} = \{\boldsymbol{I}^{D_i}, \ldots, \boldsymbol{I}^{D_p}\}$ is a non-empty set of attribute instances organized according to dimensions. $\forall \boldsymbol{I}^{D_x} \in \boldsymbol{I}^{R_i}$, $\boldsymbol{I}^{D_x}$ refers to the attribute instances on the dimension D_x;
- $f^{R_i} : \boldsymbol{V}^{R_i} \longrightarrow \boldsymbol{I}^{R_i}$ is a function associating each n-tuple of measure values $\{v^{m_1}, \ldots, v^{m_f}\}$ to one n-tuple of attribute instances $\{i^{a_1}, \ldots, i^{a_e}\}$, where $\forall v^{m_x} \in \{v^{m_1}, \ldots, v^{m_f}\}$, v^{m_x} is a value of the measure m_x (i.e. $v^{m_x} \in val(m_x)$) and $\forall i^{a_y} \in \{i^{a_1}, \ldots, i^{a_e}\}$, i^{a_y} is an instance of the attribute a_y (i.e. $i^{a_y} \in dom(a_y)$). In particular, when a result contains only attribute instances without any measure value (i.e. $\mathbf{V}^{R_i} = \varnothing$), f^{R_i} is an *empty function*.

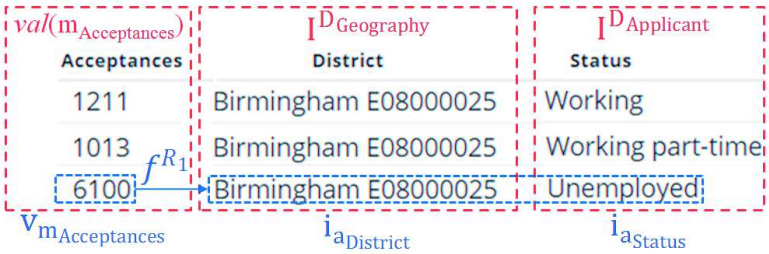


Figure 10: An example of query result

Example. Figure 10 shows the number of accepted applications for social housing by district and economic status (i.e. the result of the second generated query in appendix D).

Based on the generic modeling of result, we propose a process whose goal is to generate an overall result at the end of an analysis (cf. algorithm 3). This is done by blending together data from one or several query results. When one single query result is obtained ($|\boldsymbol{R}| = 1$), it is considered as the final analysis result (cf. lines 1 and 2). When several query results are produced, they are combined together by reference to the related instances ($\boldsymbol{T} = \{T_1, ..., T_k\}$) stored in the table of correspondences (cf. lines 3 - 22). In the following, we provide more details about how multiple query results are fused together to form one analysis result at the output of the algorithm.

Algorithm 3: Automatic generation of Analysis Result

input : A set of chosen attributes $\boldsymbol{A}$; a set of query results returned from sources $\boldsymbol{R} = \{R_1, \ldots, R_n\}$.
output: A single analysis result $R_{Analysis}$

if $|\boldsymbol{R}|=1$ **then**
  $R_{Analysis} = \boldsymbol{R}$;
else
  Find a query result R_x, such as $R_x \in \boldsymbol{R} \wedge \boldsymbol{V}^{R_x} \neq \varnothing$;
  Create a temporary result R_{temp}, *set* $R_{temp} \longleftarrow R_x$;
  Get the set of tuples $\boldsymbol{T} = \{T_1, \ldots, T_k\}$ *in the table of correspondences*;
  for *each result* $R_i \in \boldsymbol{R} \wedge R_i \neq R_x$ **do**
    for *each set of attribute instance* $\boldsymbol{I}^{D_x}$ *on the dimension* D_x *in result* R_i, $\boldsymbol{I}^{D_x} \subseteq \boldsymbol{I}^{R_i}$ **do**
      Find the set of attribute instances $\boldsymbol{I}^{D_x}_{bis}$ *on the same dimension* D_x *in* $\boldsymbol{I}^{R_{temp}}$;
      for *each pair of instances* i^{a_p} *and* i^{a_q}, *such as* $i^{a_p} \in \boldsymbol{I}^{D_x} \wedge i^{a_q} \in \boldsymbol{I}^{D_x}_{bis}$ $\wedge \exists id \in \mathbb{N}, \exists score \in \mathbb{R}$, *there exists a tuple in the table of correspondances, such as* $\{id, a_p.getNameA(), i^{a_p}, a_q.getNameA(), i^{a_q}, score\} \in \boldsymbol{T}$ **do**
        Associate i^{a_p} with i^{a_q} in R_{temp};
        if $\boldsymbol{V}^{R_i} \neq \varnothing$ **then**
          Create $f^{R_{temp}}$ *between* $\boldsymbol{V}^{R_i} \cup \boldsymbol{V}^{R_{temp}}$ *and* $\boldsymbol{I}^{D_x} \cup \boldsymbol{I}^{D_x}_{bis}$;
        else
          Create $f^{R_{temp}}$ *between* $\boldsymbol{V}^{R_{temp}}$ *and* $\boldsymbol{I}^{D_x} \cup \boldsymbol{I}^{D_x}_{bis}$;
        end
      end
    end
  end
  Aggregate R_{temp} *according to the chosen attributes* $\boldsymbol{A}$;
  $R_{Analysis} \longleftarrow R_{temp}$;
end

Specifically, the algorithm first creates a temporary result R_{temp} based on one query result containing some measure values (cf. lines 4 and 5). Then, each n-tuple of attribute instances in R_{temp} is

associated with a related n-tuple in query result R_i. This is done by referring to tuples in the table of correspondences (cf. lines 6 - 11). Meanwhile, if R_i includes some measures, related measure values from R_{temp} and R_i are grouped together and linked to the corresponding attribute instances in R_{temp} (cf. lines 12 and 13). If no measure is involved in R_i, only measure values from R_{temp} are associated with related attribute instances (cf. line 15). At last, a single analysis result is generated by aggregating the measures in R_{temp} according to the attributes specified by a decision-maker (cf. lines 20 and 21). Note that the abstract notation in algorithm 3 follows the conceptual *Unified Cube* modeling presented in sections 3, while the operations correspond to those in the metamodel described in section 4.1.

Example. To explain the execution of the algorithm 3, we illustrate how the analysis result is generated for the most complex analysis (the fifth analysis) in table 2. Remember that three queries were generated for this analysis: the first query R_1 aggregates the sum of $m_{Applications}$ by $a_{Housing\_District}$ and $a_{Applicant\_Status}$ from the DW; the second query R_2 aggregates the sum of $m_{Acceptances}$ by $a_{District}$ and a_{Status} from the LOD1 dataset; the third query R_3 associates each district (i.e. a_{Area}) with its region (i.e. a_{Region}) from the LOD2 dataset without any measure.

After the execution of all generated queries, the algorithm 3 receives three results from different sources. It first creates a temporary result R_{temp} based on the query result R_1 from the DW (cf. figure 11(1)). Then, attribute instances from R_{temp} are grouped with those from R_2 (cf. figure 11(2)). During this step, attribute instances involved in correlative mappings are grouped together by referring to the table of correspondences (cf. lines 6 - 11 in algorithm 3). Next, measure values from R_{temp} are associated with related ones from R_2 (cf. figure 11(3)).

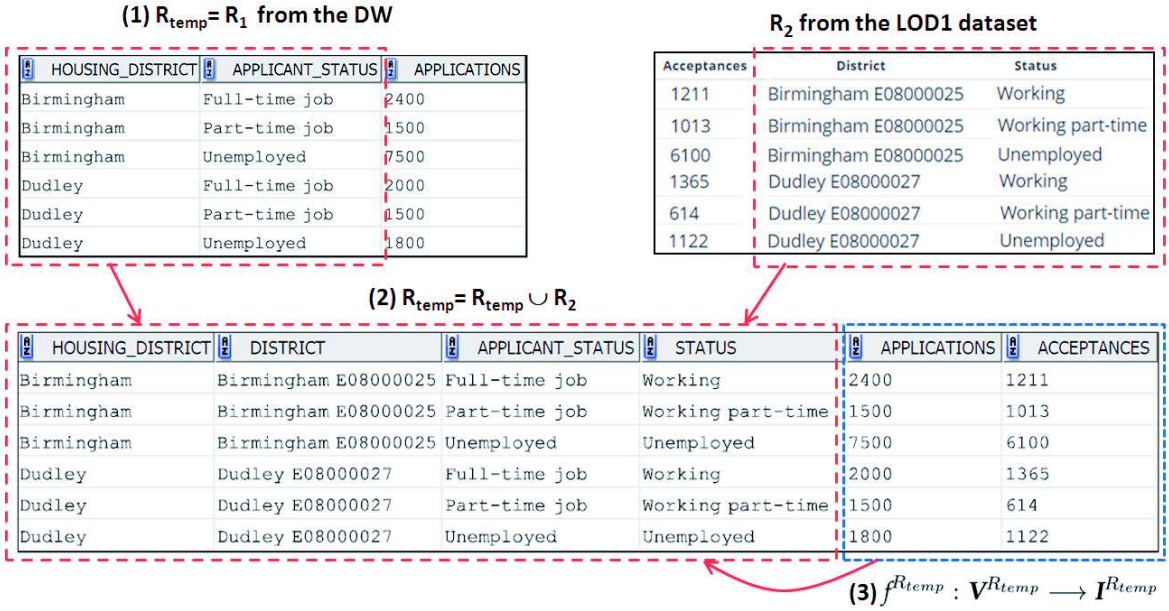
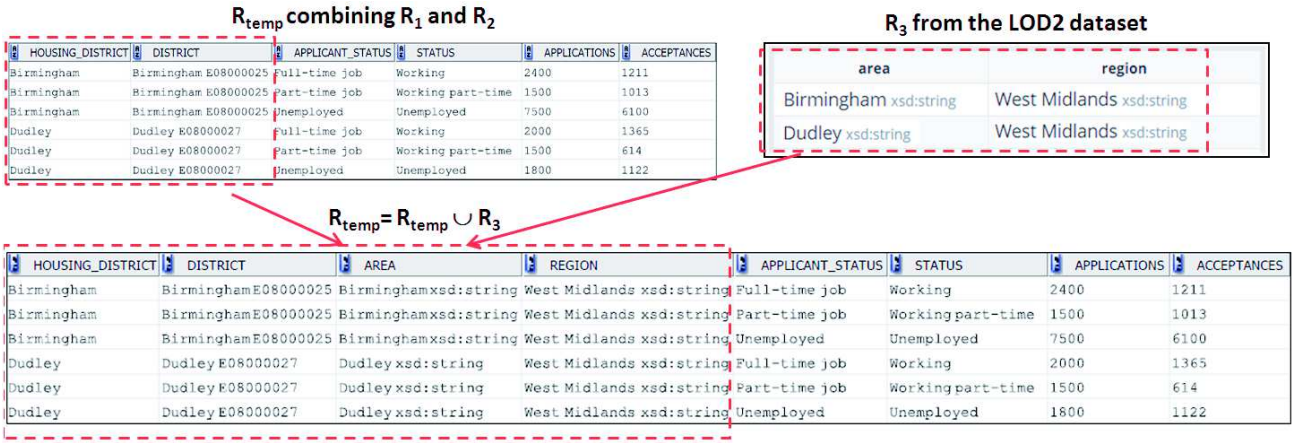


Figure 11: Combining query results from the DW and the LOD1 dataset

Query result R_3 does not contain any measure. It provides a complementary level of the geographical dimension in R_{temp}. To merge R_3 with R_{temp}, the algorithm refers to the table of correspondences

to associate instances of $a_{District}$ with those of a_{Area}. The instances of a_{Region} are merged into new R_{temp} along with corresponding areas (cf. figure 12).



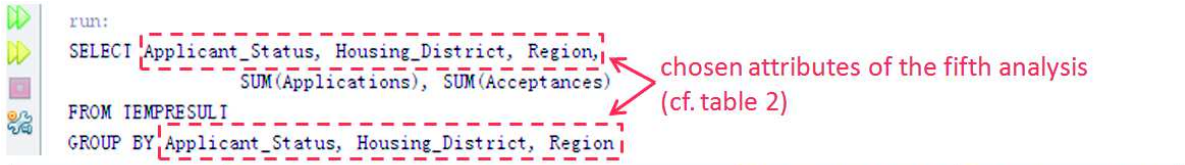
HOUSING_DISTRICT	DISTRICT	APPLICANT_STATUS	STATUS	APPLICATIONS	ACCEPTANCES
Birmingham	Birmingham E08000025	Full-time job	Working	2400	1211
Birmingham	Birmingham E08000025	Part-time job	Working part-time	1500	1013
Birmingham	Birmingham E08000025	Unemployed	Unemployed	7500	6100
Dudley	Dudley E08000027	Full-time job	Working	2000	1365
Dudley	Dudley E08000027	Part-time job	Working part-time	1500	614
Dudley	Dudley E08000027	Unemployed	Unemployed	1800	1122

area	region
Birmingham xsd:string	West Midlands xsd:string
Dudley xsd:string	West Midlands xsd:string

HOUSING_DISTRICT	DISTRICT	AREA	REGION	APPLICANT_STATUS	STATUS	APPLICATIONS	ACCEPTANCES
Birmingham	Birmingham E08000025	Birmingham xsd:string	West Midlands xsd:string	Full-time job	Working	2400	1211
Birmingham	Birmingham E08000025	Birmingham xsd:string	West Midlands xsd:string	Part-time job	Working part-time	1500	1013
Birmingham	Birmingham E08000025	Birmingham xsd:string	West Midlands xsd:string	Unemployed	Unemployed	7500	6100
Dudley	Dudley E08000027	Dudley xsd:string	West Midlands xsd:string	Full-time job	Working	2000	1365
Dudley	Dudley E08000027	Dudley xsd:string	West Midlands xsd:string	Part-time job	Working part-time	1500	614
Dudley	Dudley E08000027	Dudley xsd:string	West Midlands xsd:string	Unemployed	Unemployed	1800	1122

Figure 12: Combining R_{temp} with query result from the LOD2 dataset

The final analysis result is generated by aggregating measures according to attributes chosen by a decision-maker. In our example, the attributes $a_{Applicant\_Status}$, $a_{Housing\_District}$ and a_{Region} consists of the chosen attributes of the analysis (cf. table 2), the algorithm automatically generates a query aggregating the temporary result according to $a_{Applicant\_Status}$, $a_{Housing\_District}$ and a_{Region} (cf. lines 19 and 20 in algorithm 3). The final analysis result is shown in figure 13.



APPLICANT_STATUS	HOUSING_DISTRICT	REGION	SUM(APPLICATIONS)	SUM(ACCEPTANCES)
Full-time job	Birmingham	West Midlands xsd:string	2400	1211
Unemployed	Birmingham	West Midlands xsd:string	7500	6100
Unemployed	Dudley	West Midlands xsd:string	1800	1122
Full-time job	Dudley	West Midlands xsd:string	2000	1365
Part-time job	Birmingham	West Midlands xsd:string	1500	1013
Part-time job	Dudley	West Midlands xsd:string	1500	614

Figure 13: Analysis result about number of submitted and accepted applications by applicant's status and housing district, region

The advantages of our proposed decision-support process are twofold. First, our proposed decision-support process always provides up-to-date data to decision-makers. Warehoused data and LOD are queried on-the-fly during analyses without materializing them in a stationary repository. Second, our proposed decision-support process facilitates decision-makers analysis tasks. An analysis result

based on data from various sources is automatically created in an on-demand manner. In this way, the distribution of data throughout multiple sources, the different schemas of various sources, and the complex querying languages with specific syntax are all hidden from end users by our proposed decision-support process.

5.3. Multisource analysis framework

To enable analyses on multiple sources in a user-friendly manner, we develop a multisource analysis framework which present only business-oriented concepts to decision-makers during analyses. The overall architecture of the multisource analysis framework is shown in figure 14.

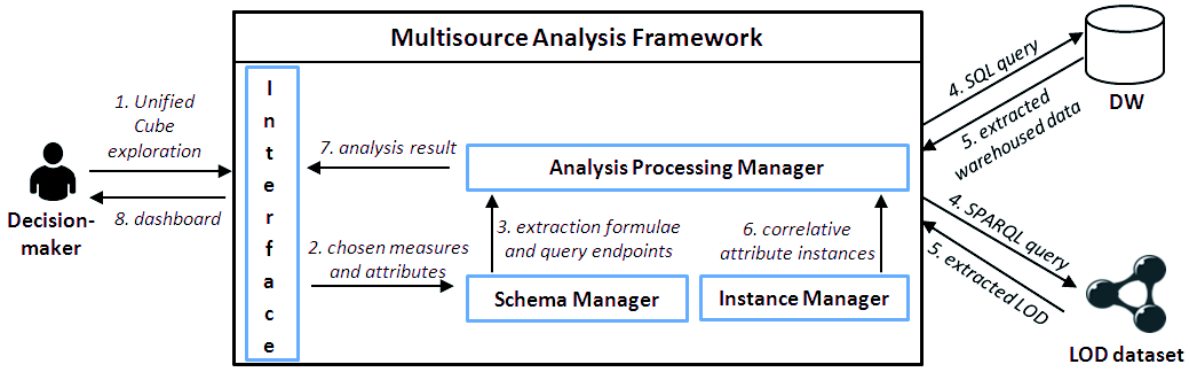


Figure 14: Architecture of the multisource analysis framework

Specifically, a *Unified Cube* is implemented at both the schema and the instance levels by *schema manager* and *instance manager* respectively. The *schema manager* deals with concepts related to the unified view of multiple data sources. It is made up of (i) a metamodel implemented in the Oracle DBMS and (ii) a Java program instantiating the metamodel. The *instance manager* deals with correlative attribute instances from different sources. The Oracle DBMS hosts a relational table of correspondences in the *instance manager*. A Java program implements our instance matching process and populates the relational table of correspondences. Note that a *Unified Cube* can be built upon any source which provides a querying endpoint (e.g., an internal DW fully controlled by an organization, an external DW hosted by a business partner, an online LOD dataset, ...). Moreover, it is possible to populate the table of correspondences on-the-fly and keep it in memory without materialization. However, to reduce analysis runtime, we choose to populate the table of correspondences prior to querying and materialize it in a relational DBMS.

The analysis processing methods are implemented through two Java programs. They allow (i) generating queries to extract data from multiple sources and (ii) integrating extracted data into one analysis result. Both Java programs are included in the *analysis processing manager*. Note that queries discussed in section 5.1 (cf. appendixes) and analysis results presented in section 5.2 are automatically generated by the *analysis processing manager* of our multisource analysis framework.

We develop a graphical *interface* by extending the ones of our previously proposed analysis tools [35, 34]. The *interface* aims at facilitating interactions between a decision-maker and the multisource

analysis framework. As shown in figure 15, a decision-maker can express an analysis need by choosing measures and attributes within related levels through the graphical notation of *Unified Cube*s (cf. upper part of the graphical *interface*). At the end of an analysis, a decision-maker can visualize an analysis result in tabular and graphic forms (cf. lower part of the graphical *interface*).

Figure 15: Graphical interface of the multisource analysis framework

Our proposed analysis framework supports a user-friendly decision-making process in our analysis framework. As shown in figure 14, a decision-maker starts an analysis by exploring a *Unified Cube* in graphical notation through an interface (cf. arrow 1). She/he clicks on measures and attributes related to her/his analysis needs. The interface sends the chosen measures and attributes to the *schema manager* (cf. arrow 2). The latter looks up in the metamodel to find extraction formulae and query endpoints of the chosen measures and attributes (cf. arrow 3). Next, the *analysis processing manager* generates a set of queries and sends them to the corresponding query endpoints of different sources. Each query includes measures and attributes from one source (cf. arrows 4). After queries execution, the *analysis processing manager* receives data extracted from different sources (cf. arrows 5). It then refers to the correlative relationships in the *instance manager* to associate equivalent attribute instances together among extracted data (cf. arrow 6). One analysis result is generated by the *analysis processing manager* based on the chosen measures and attributes (cf. arrow 7). At last, the interface provides the decision-maker with a dashboard representing the analysis result in tabular and graphic forms (cf. arrow 8). In this way, decision-makers only interact with some business-oriented concepts during analyses. The analysis processing is completely hidden from decision-makers.

6. Experimental assessments

To enable analyses on data from multiple sources, one key step consists of unifying data extracted from different sources together to form one unique analysis result. This unification is based on the table of correspondences which is populated through an instance matching process (cf. figure 6). By exclusively using string similarity measures, our proposed matching process is generic enough to identify correlative instances without formally described data semantics. To study the feasibility and efficiency of our proposed matching process, we carry out some experimental assessments.

In this section, we first describe the inputs of our experimental assessments composed of two data collections and sixteen string similarity measures (cf. section 6.1). Second, we present the results of our experimental assessments. We also discuss the influences of four matching setups and different features of data collections on matching efficacy and efficiency (cf. section 6.2). Third, based on the experimental results, we propose some generic guidelines for efficient use of string similarity measures to match correlative instances in a *Unified Cube* (cf. section 6.3). At last, we validate our proposed guidelines by using benchmarks of *Ontology Alignment Evaluation Initiative*[14] (cf. section 6.4).

6.1. Input

During our experimental assessments, we use two collections of real-world data. Each collection covers a specific domain and has distinctive features.

The first collection is named *GeoData*. It contains geographic data in the RDF format published by the UK Department for Communities and Local Government (CLG)[15]. The CLG provides multidimensional LOD datasets related to social housing, taxation of local governments, etc. Most CLG datasets contain a geographic dimension composed of one level named *District*. The geographic dimension can be completed by adding more levels such as *County* and *Region* published by the Office for National Statistics (ONS)[16]. To do so, each *District* from the CLG dataset is associated with a corresponding *District* from the ONS dataset through the property `owl:sameAs`. These correspondences are used as *baselines*, i.e. a set of actual mappings between instances from two datasets. The objective is to match 237 *non-metropolitan districts* from the CLG dataset with 280 *non-metropolitan and metropolitan districts* from the ONS dataset. The percentage of mismatches is $(280 - 237)/237 = 18.14\%$. The CLG dataset includes 25 079 RDF triples, while the ONS dataset contains only 4256 RDF triplet. Descriptive information about districts, such as name and type, is relatively short. The average string length is 24.6 characters including data type descriptors and name spaces.

The second collection is named *DBLP*. It contains bibliographic data related to scientific publications provided by DBLP (e.g. conference papers, journal articles, etc.). Two different implementations of the bibliographic data are managed by the *European Network of Excellence ReSIST*[17] and the *L3S Research Center*[18]. The *baselines* consist of a set of `owl:sameAs` properties which links each publication hosted by *ReSIST* to a corresponding publication hosted by *L3S*. We extract 1000 publications

[14] http://oaei.ontologymatching.org/
[15] http://opendatacommunities.org/
[16] http://statistics.data.gov.uk/
[17] http://dblp.rkbexplorer.com/sparql
[18] http://dblp.l3s.de/d2r/sparql.

from the *ReSIST* and the *L3S* datasets. Each extracted publication from *ReSIST* has one unique equivalent publication extracted from *L3S*. We then introduce different percents of *mismatches* in extracted data by adding in one dataset some publications which have no equivalent one in the other dataset. Six scale factors of mismatches are proposed to include from 0% to 50% mismatches in both the ReSIST and the L3S datasets. The objective is to correctly find the 1000 corresponding publications. The ReSIST datasets include from 16983 to 25512 triples, while the L3S datasets contain from 23549 to 35344 triples. Descriptive information about publications, such as title and conference name, is composed of long strings. The average string length is about 290 characters including data type descriptors and name spaces.

A summary of the datasets used in our experimental assessments is provided in table 3.

Table 3: Details of the datasets used in our experimental assessments

Title	**Datasets**		**Mismatch**	**Expected Matching**	**Avg Length**
	Name	***Volume***			
GeoData	CLG	25079 triples	18.14%	237 pairs	24.6
	ONS	4256 triples			
DBLP0	ReSIST	16983 triples	0%	1000 pairs	290
	L3S	23549 triples			
DBLP10	ReSIST	18680 triples	10%	1000 pairs	
	L3S	25819 triples			
DBLP20	ReSIST	20423 triples	20%	1000 pairs	
	L3S	28100 triples			
DBLP30	ReSIST	22069 triples	30%	1000 pairs	
	L3S	30422 triples			
DBLP40	ReSIST	23778 triples	40%	1000 pairs	
	L3S	32900 triples			
DBLP50	ReSIST	25512 triples	50%	1000 pairs	
	L3S	35344 triples			

We find out sixteen generic and widely-used string similarity measures in the scientific literature. Based on the definition, we classify the similarity measures into six groups (cf. table 4).

6.2. Protocole, observations and discussions

6.2.1. Protocol

The objective of the experimental assessments is to find out if string similarity measures can be used to match correlative attribute instances. And if so, how and when they should be used to maximize their efficiency during the matching. To do so, we must answer the following questions.

Table 4: Sixteen string similarity measures according to six groups

Similarity Measure		Description	Group
1	Soundex	It is based on phonetic encoding to match homophones	Phonetic
2	N Grams	It compares subsequences of N characters between strings	Subsequence based
3	Levenshtein	One of the most widely used string similarity measures based on edit distance, i.e. copy, substitute, insert and delete a character from one string to another	Edit Distance
4	Needleman Wunch	A weighted variant of Levenshtein by adding a variable cost to the gap, i.e. insert and delete	
5	Smith Waterman	A variant of Needleman Wunch which is originally developed to identify optimal matching between related DNA and protein sequences	
6	Smith Waterman Gotoh	An extension of Smith Waterman by adding a scoring system of gap penalty to align small portion of genetic codes	
7	Monge Elkan	An extension of Smith Waterman by allowing a specific gap penalty function between sequences	
8	Jaro	A linear sum assignment variant of edit distance originally developed in the field of record linkage	Jaro
9	JaroWinkler	A variant of Jaro suited for short strings	
10	Block Distance	A similarity measure based on n-dimensional vector space defined through the characters of input strings. The similarity is calculated by summing the edge distances	Characters based
11	Cosine Similarity	Instead of summing the edge distances, this variant calculates the cosine value of the angle between input strings	
12	Matching Coefficient	A basic vector-based string similarity measures comparing terms (or sets of characters) within strings. The position of the terms is not taken into account	Term based
13	Dice Similarity	Also known as F1 score, it is defined as twice the number of common terms divided by the total number of terms	
14	Overlap Coefficient	It is defined as the size of common terms divided by the size of the shortest input string	
15	Jaccard	It is defined as the size of the intersection divided by the size of the union of terms within the input strings	
16	Euclidean Distance	The similarity is measured through the length of the line segment between two vectors composed of string terms	

- Do string similarity measures of the same group have similar efficiency when being applied to the same data sources?
- Does each group of string similarity measures keeps the same efficiency when being applied to different data sources?

- What are the influence factors on the efficiency of string similarity measures?

The matching candidates are formed according to four combinations of matching setups, namely Concatenated&Unprocessed, Concatenated&Optimized, Separated&Unprocessed, and Separated& Optimized (cf. section 4.2). The efficiency of each string similarity measure is evaluated through the F-measure and the runtime. Specifically, we compare the result obtained by each string similarity measure with baselines to calculate the precision and recall scores of the matching. The precision is the ratio of the number of true positives to the retrieved mappings, while the recall is the ratio of the number of true positives to the actual mappings expected. The F-measure is calculated for each string similarity measure by combining the precision and the recall, such as: $F - measure = 2 \times precision \times recall \div (precision + recall)$. The runtime consists of the CPU time (in seconds) recorded during the execution of each string similarity measure. A similarity measure is efficient if it produces a *high* F-measure within a *short* time.

All similarity measures are implemented within an open source Java library[19]. Developed by UK Sheffield University, this Java library is widely used by both software developers and matching campaigns organized by *Ontology Alignment Evaluation Initiative*. The hardware configuration of the multi-threads execution environment is as follows: CPU of two AMD Opteron 6262HE with 16 cores, RAM of 128 GB and SAS 10K disk.

6.2.2. Observations and discussions

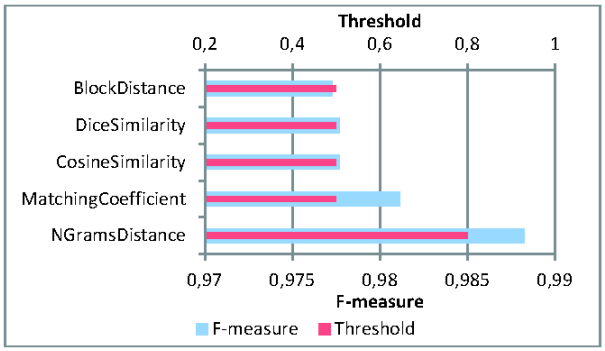
The first tests consist of comparing the F-measure of string similarity measures according to different influence factors.

Influences of matching setups on F-measure Our first objective is to study if the F-measure of a string similarity measure changes with different matching setups. More specifically, we intend to study:

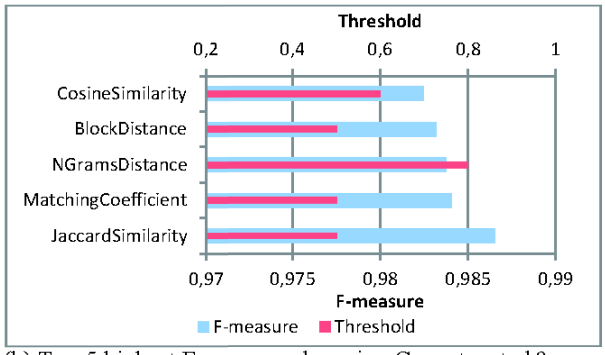
- if the best F-measure of each similarity measure varies according to different matching setups;
- if the threshold producing the best F-measure (so-called *optimal threshold*) for a given similarity measure changes with matching setups;
- if the similarity measure producing the highest F-measure for a given dataset is the same according to different matching setups;
- if the value of the highest F-measure obtained in a dataset changes with matching setups.

With these questions, we record the best F-measure and the corresponding optimal threshold of each string similarity measure executed in geographical and bibliographic datasets (cf. table 3). We notice that matching results of bibliographic datasets are similar to each other. For simplicity reason and due to limited space, we present result obtained in the largest bibliographic datasets (i.e. DBLP50). Figure 16 shows the top 5 similarity measures producing the highest F-measure respectively in DBLP50 and GeoData datasets.

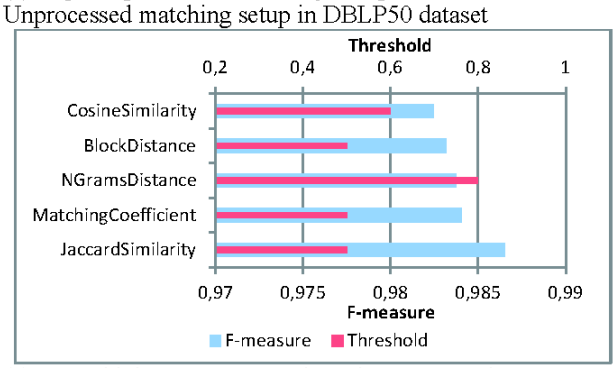
[19] https://sourceforge.net/projects/simmetrics/



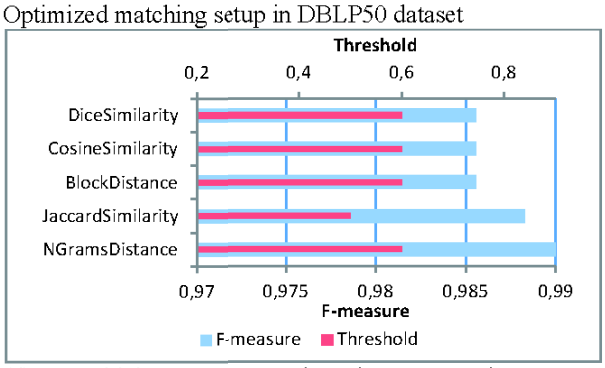
(a) Top 5 highest F-measures by using Concatenated& Unprocessed matching setup in DBLP50 dataset



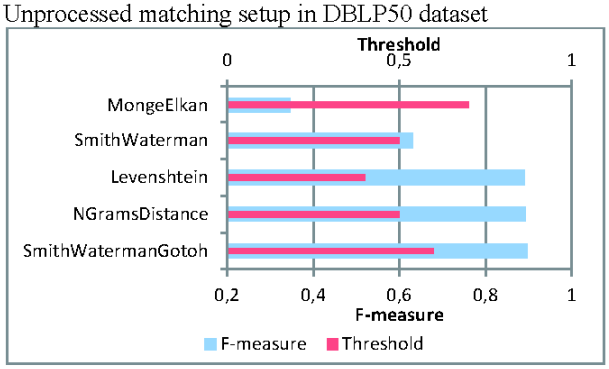
(b) Top 5 highest F-measures by using Concatenated& Optimized matching setup in DBLP50 dataset



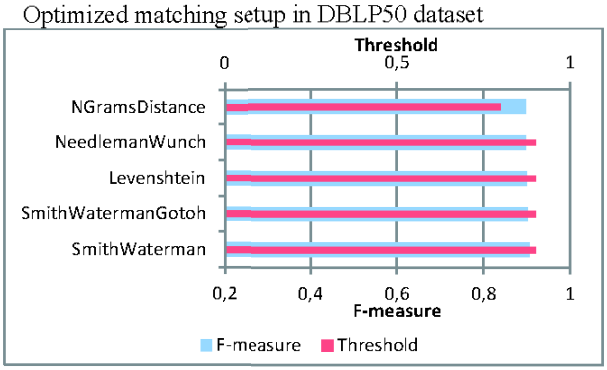
(c) Top 5 highest F-measures by using Separated& Unprocessed matching setup in DBLP50 dataset



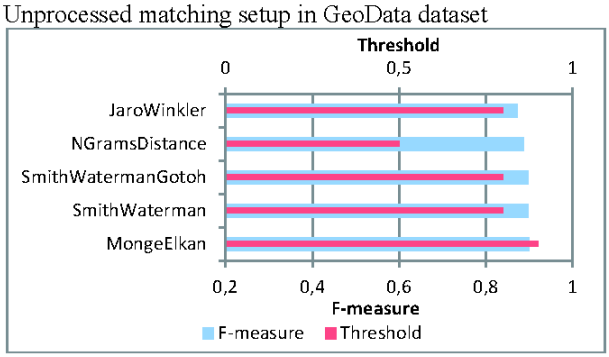
(d) Top 5 highest F-measures by using Separated& Optimized matching setup in DBLP50 dataset



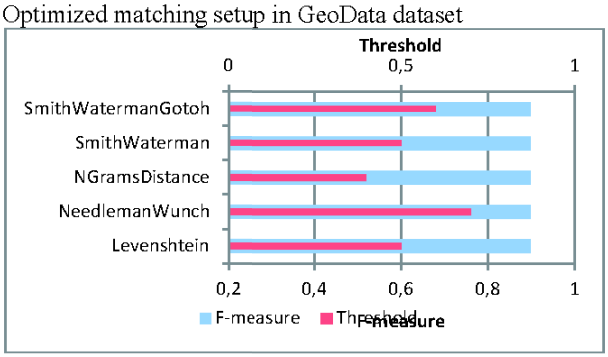
(e) Top 5 highest F-measures by using Concatenated& Unprocessed matching setup in GeoData dataset



(f) Top 5 highest F-measures by using Concatenated& Optimized matching setup in GeoData dataset



(g) Top 5 highest F-measures by using Separated& Unprocessed matching setup in GeoData dataset



(h) Top 5 highest F-measures by using Separated& Optimized matching setup in GeoData dataset

Figure 16: Top 5 F-measures with the corresponding optimal thresholds according to matching setup

Observations From figure 16, we can see that the best F-measure of a similarity measure for a given dataset varies according to different matching setups: (i) some similarity measures are insensitive to different matching setups, e.g. the best F-measure of *N Gram Distance* varies only 1% in both DBLP50 and GeoData datasets; (ii) some similarity measures are very sensitive to matching setups in some datasets, e.g. compared to the Concatenated&Unprocessed matching setup, the Separated&Optimized matching setup allows the best F-measure of the *Smith Waterman* measure to increase by about 45%.

The optimal threshold of each similarity measure is influenced by matching setups. The changes do not follow obvious rules. For instances, the Optimized matching setups increase the optimal threshold of *N Gram Distance* by 60% compared to the Unprocessed matching setups, while the optimal thresholds of *Smith Waterman* decrease when matching candidates get too long or too short (i.e. according to the Concatenated&Unprocessed and the Separated&Optimized matching setups).

In both figures, the highest F-measure of all similarity measures does not vary much according to different matching setups. For instance, in the DBLP50 dataset, the highest best F-measure (i.e. 0.99) is obtained by the *N Gram Distance* measure through the Separated&Optimized matching setup, while the "lowest" best F-measure (i.e. 0.986) is produced by *Jaccard Similarity* when the Concatenated&Unprocessed is used.

Conclusion Based on the previous observations, we can conclude the matching setup is not a negligible factor during matching processes. The performance of some similarity measures depends heavily on the involved matching setup. However, the matching setup does not consist of a determinant factor for the matching in a dataset. By coupling an appropriate similarity measure with an optimal threshold, all matching setups allow producing almost the same matching results for a given dataset.

Appropriate matching setups for each string similarity measure groups Our second objective is to find out the most appropriate matching setup allowing maximizing the F-measure for string similarity measures in each group. To do so, we study:

- if the string similarity measures within a group obtain similar F-measures when the same matching setup is applied;
- what are the matching setups allowing the string similarity measures within a group to produce the highest F-measure.

To answer these questions, we apply all similarity measures to all datasets according to different matching setups. For each similarity measure, we record the best F-measure obtained with the optimal threshold in each dataset. Since the same trend is found in all results and due to limited space, we only present the results obtained in the largest dataset (i.e. DBLP50).

Observations At first glance, not all matching setups have the same impact on the F-measure of all groups of similarity measures (cf. figure 17).

The Phonetic similarity measure (i.e. *Soundex*) produces an acceptable F-measure only when the matching setup Separated&Optimized is applied. The Concatenated&Unprocessed matching setup is the least suitable for *Soundex* (with a F-measure close to zero). The same situation happens in the group of Jaro (i.e. *Jaro* and his extension *Jaro Winkler*): even though they produce an acceptable

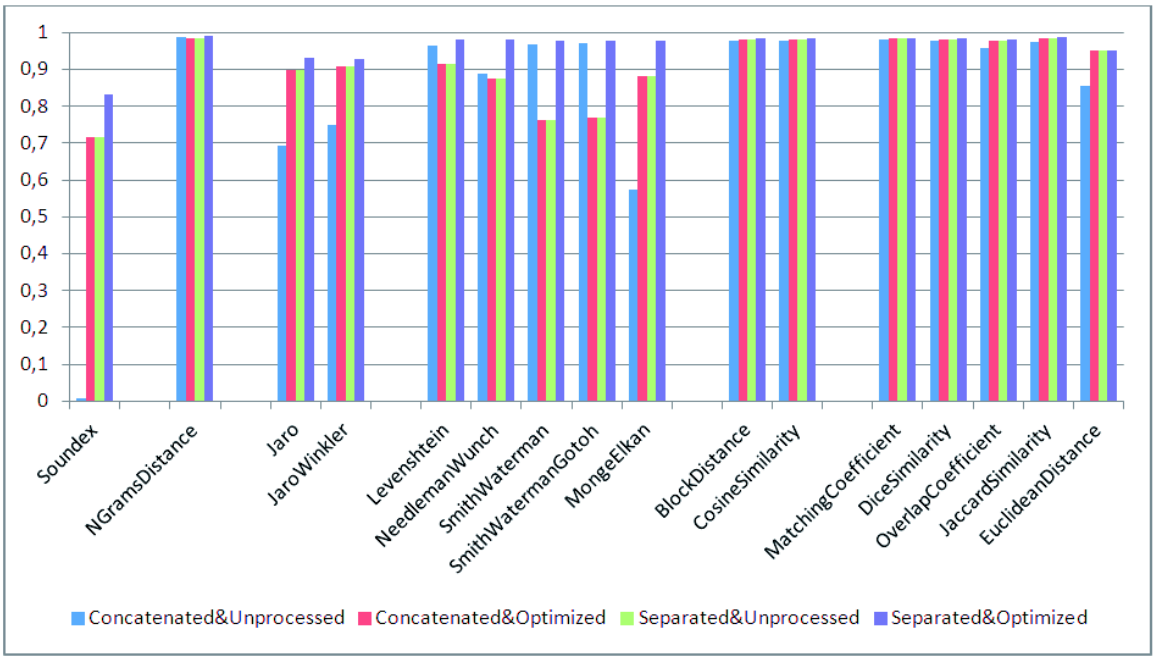


Figure 17: Highest F-measure according to similarity measuresgroups and matching setups

F-measure in the case of the Concatenated&Unprocessed matching setup, The Separated and the Optimized matching setups increase significantly the F-measure of the similarity measures within Jaro group.

Similarity measure based on subsequences of strings (i.e. *N Grams Distance*) is insensitive to matching setups, the same goes for most Term-based similarity measures (i.e. *Dice Similarity*, *Euclidean Distance*, *Jaccard Similarity*, *Matching Coefficient* and *Overlap Coefficient*) and Character-based similarity measures (i.e. *Block Distance* and *Cosine Similarity*). They are also among the most efficient similarity measures with an average F-measure around 0.98.

Three similarity measures based on Edit Distance (i.e. *Levenshtein*, *Smith Waterman* and *Smith Waterman Gotoh*) go to extremes: to produce the highest F-measure, multiple attribute instances should be matched in the form of (i) long strings produced by the Concatenated&Unprocessed matching setup or (ii) short strings produced by the Separated&Optimized matching setup. The similarity measures *Monge Elkan* and *Needleman Wunch*, however, are only suitable for the Separated&Optimized matching setup. They obtain relatively low F-measures in the case of the Concatenated&Unprocessed matching setup due to the following reasons:

- Comparing to the basic *Levenshtein* measure, *Monge Elkan* gives a relatively low cost to a sequence of character insertions and character deletions. It tends to ”ignore” the differences between shorter strings, which produces a lower F-measure in the case of the Concatenated& Unprocessed matching setup;

- Comparing to the basic *Levenshtein* measure, *Needleman Wunch* penalizes more insertions and deletions by affecting a higher cost. As a result, it is more appropriate to identify perfect

matches between strings. The highest F-measure is slightly lower in the case of the Concatenated&Unprocessed matching setup, since perfect matches are less likely to exist among long strings after concatenation of unprocessed attribute instances.

Conclusion Based on the previous observations, we can conclude string similarity measures in the same group produce almost the same F-measure when the same matching setup is used. There is at least one matching setup that allows maximizing the F-measure of all similarity measures within each group.

Influences of datasets on F-measure of string similarity measures Our third objective consists of coupling a dataset with the most appropriate similarity measures. To do so, we study:

- if attribute instances' length within a dataset has influences on similarity measures' F-measure;
- if similarity measures' F-measure varies with the proportion of mismatches within datasets.

To answer these questions, we apply all sixteen string similarity measures to all geographical and bibliographic datasets. We record the highest F-measure of each similarity measure when the optimal threshold and the most suitable matching setup are applied. Figure 18 shows the F-measure of each similarity measure according to their group.

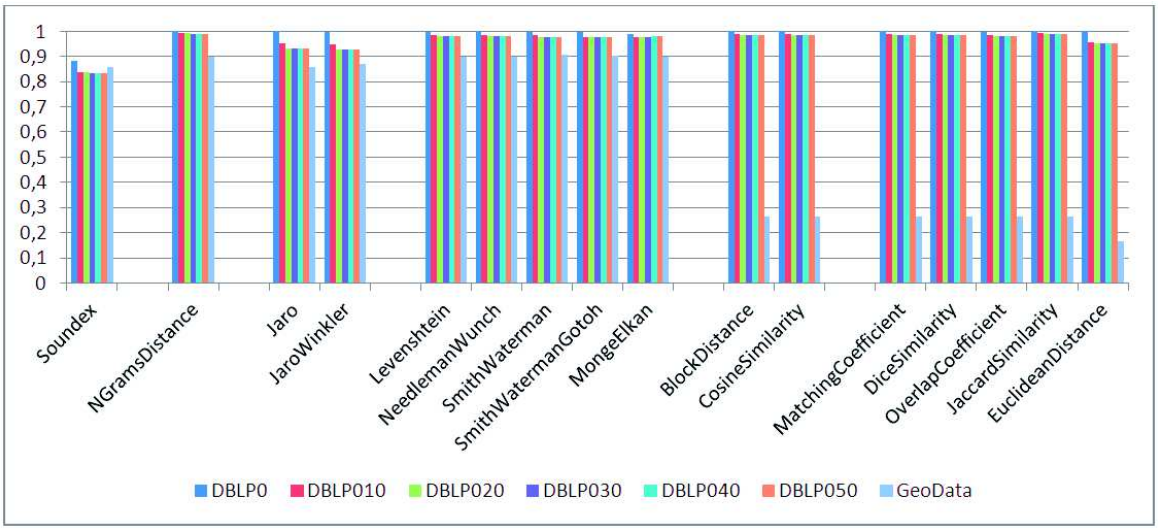


Figure 18: F-measure produced by each similarity measure in different datasets

We can see string similarity measures within the same group yield almost the same F-measures in the same dataset when the optimal threshold and the most suitable matching setup are used. From geographical GeoData dataset to bibliographic DBLP datasets, the average length of string becomes longer (cf. table 3). The F-measure of string similarity measures changes accordingly: (i) similarity measures within Characters based group (e.g. *Block Distance*) and the Term based group (e.g. *Matching Coefficient*) are more suitable for matching longer strings (e.g. DBLP50) than short strings (e.g.

GeoData); (ii) similarity measures within the other groups cope well with both long and short strings. The length of strings has the least influence on the Phonetic similarity measure *Soundex*, yet its highest F-measure is not the highest among all similarity measures. The *N Grams Distance* similarity measure has a slight preference for shorter strings. But its overall F-measure, whose highest value is about 0.99, outperforms the other similarity measures.

We can also observe from DBLP0 to DBLP50 the highest F-measure of all similarity measures does not change much, even through the proportion of mismatches increases. To better understand the influences of mismatches on the F-measure of similarity measures, we calculate the standard deviation of the highest F-measure produced by each similarity measures in the six DBLP datasets.

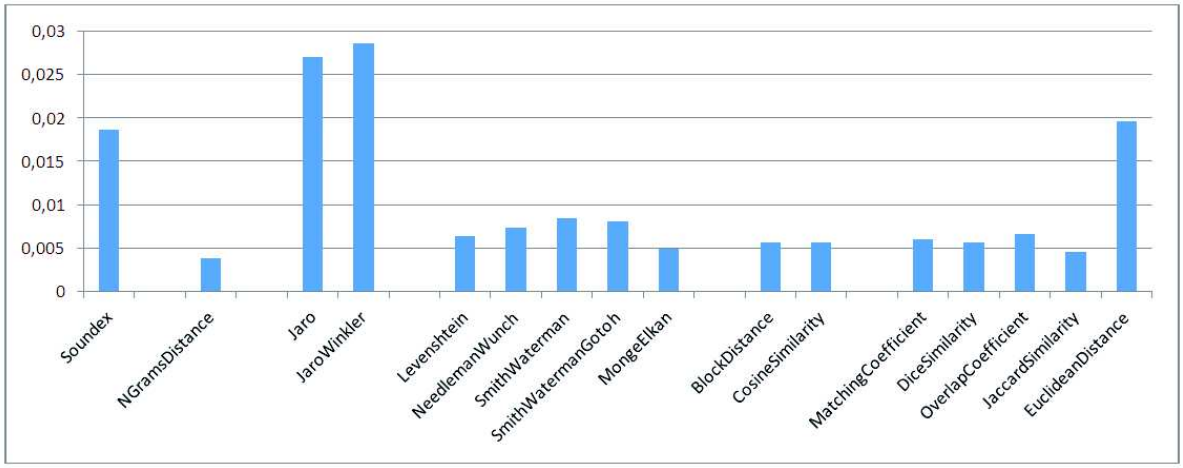


Figure 19: Standard deviation of each measures highest F-measure obtained from DBLP0 to DBLP50 datasets

From the figure 19, we can see (i) the overall standard deviation is quite low (no more than 0.03), which signifies the proportion of mismatches has little influence on the F-measure of similarity measures; (ii) the similarity measures within the same group keep almost the same F-measure when being applied to datasets with different proportions of mismatches. The similarity measures within the group *Jaro* and the *Soundex* measure are more sensitive to mismatches than the others; (iii) the only exception is *Euclidean Distance* which has a higher standard deviation than the others within Term-based group.

The last observation is in accordance with what the authors of [16] have noticed: *Euclidean Distance* is very sensitive to mismatches. It produces a perfect F-measure (i.e. F-measure=1) in DBLP0 dataset which is higher than the other similarity measures in the same group. But when mismatches are introduced in the datasets, *Euclidean Distance* obtains a lower F-measure than the others. Nevertheless, mismatches still have little influence on this similarity measure, since its standard deviation is no more than 0.02.

Conclusion Based on the previous observations, we can conclude all similarity measures manage to produce almost the same F-measure no matter how many mismatches are involved in sources.

However, not all similarity measures are suitable for all datasets: some similarity measures work well with datasets containing either long or short strings, while the others are suitable for datasets including strings of all sizes.

Efficiency of String Similarity Measures The second tests consists of identifying the most efficient similarity measures, i.e. similarity measures producing a high F-measure in a short time. Based on the results of our previous tests, we can rank similarity measures according to their F-measures. To do so, we first choose the top five similarity measures producing the highest F-measure according to four matching setups when being applied to seven datasets (cf. table 5). Other similarity measures not listed here are not studied in this section due to their low F-measure.

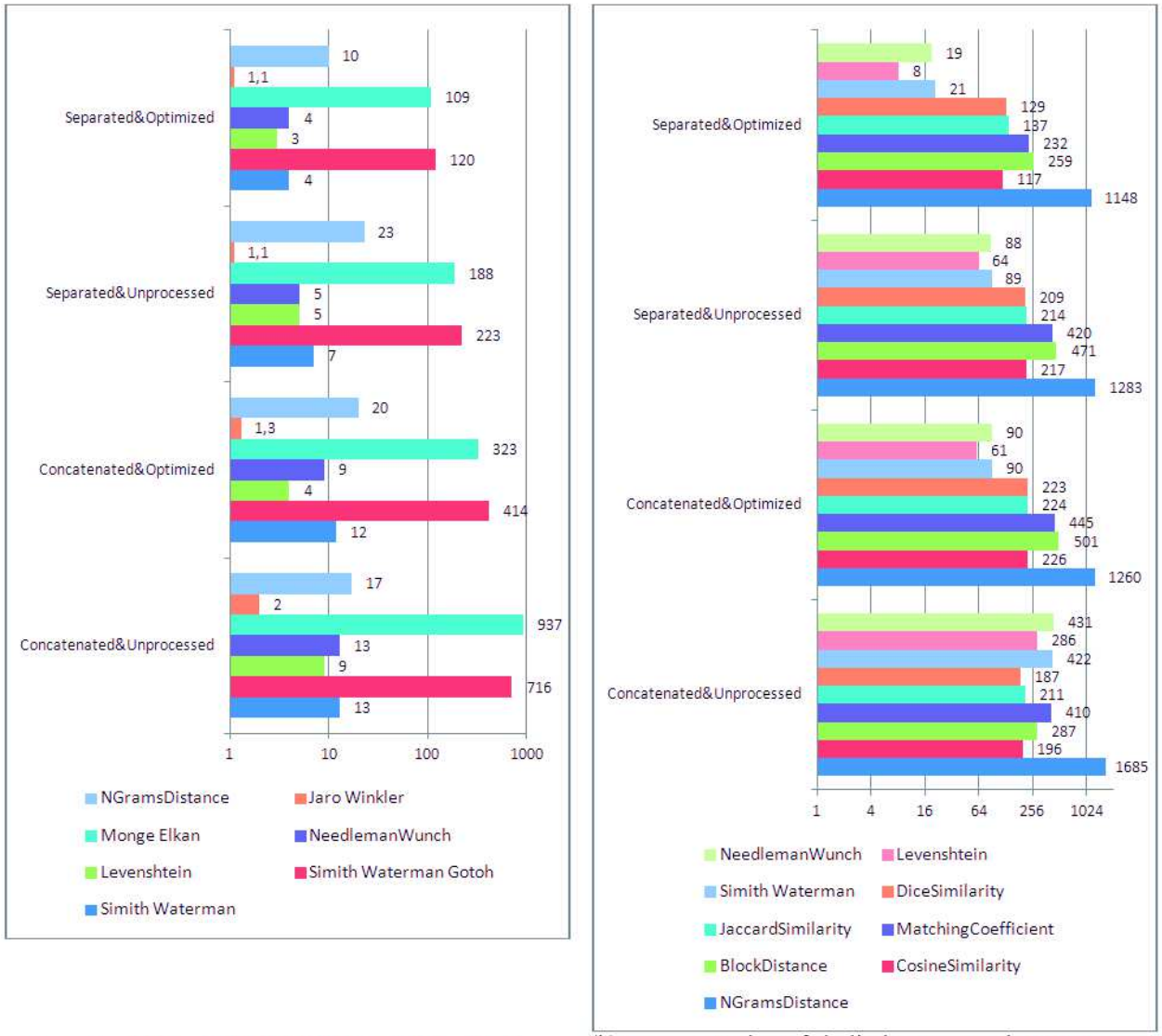
Table 5: Ranking of string similarity measures based on their occurrence among the top five ones according to matching setups

	Similarity Measures	C&U	S&U	C&O	S&O	Total
1.	N Grams Distance	7	7	7	7	28
2.	Cosine Similarity	5	6	6	6	23
3.	Block Distance	5	6	6	6	23
4.	Matching Coefficient	5	6	6	2	19
5.	Jaccard Similarity	0	5	6	6	17
6.	Smith Waterman	4	2	2	2	10
7.	Dice Similarity	4	1	0	4	9
8.	Smith Waterman Gotoh	3	2	2	2	9
9.	Levenshtein	3	2	0	2	7
10.	Needleman Wunch	2	1	0	2	5
11.	Monge Elkan	1	1	2	0	4
12.	Jaro Winkler	0	0	2	0	2

Remark. C&U = Concatenated&Unprocessed, S&U = Separated&Unprocessed, C&O = Concatenated&Optimized, S&O = Separated&Optimized

For each similarity measure in table 5, we record the runtime (CPU time in seconds) during the execution in the corresponding dataset according to different matching setups. The runtime of a similarity measure in GeoData dataset is the average runtime of five different executions (cf. figure 20(a)). For bibliographic data, we notice even if data volume increases by 225% from DBLP0 to DBLP50, the runtime of all similarity measures augments only from 5.2% to 16.9% with an average of 8% (owing to parallel computation). Due to the insignificant differences, we calculate the average runtime obtained in all six DBLP datasets (cf. figure 20(b)).

Observations From figure 20, we can see that when different matching setups are used, the gain in runtime for some similarity measures is important.



(a) Average runtime of similarity measures in GeoData dataset

(b) Average runtime of similarity measures in DBLP dataset

Figure 20: Average runtime of similarity measures in geographic and bibliographic datasets

Independently of similarity measures, the fastest matching setup is Separated&Optimized, which produces the shortest matching candidates among all matching setups. The slowest matching setup is Concatenated&Unprocessed which concatenates all unprocessed attribute instances in a long string.

The runtime of *Smith Waterman Gotoh* and *Monge Elkan* increases exponentially as the matching candidates become longer (from Separated&Optimized to Concatenated&Unprocessed). Although they are among the most effective measures, the efficiency of *Smith Waterman Gotoh* and *Monge Elkan* is low due to the time-consuming execution.

The *Jaro Winkler* measure produces matching results in less than 2 seconds in the GeoData dataset. However, since it favors short attribute instances sharing a common prefix, its efficiency in the DBLP datasets is lower than the others due to its relatively low efficacy for longer strings.

The Character based and Term based similarity measures produce a high F-measure within reasonable time in the DBLP datasets, such as *Cosine Similarity*, *Block Distance*, *Matching Coefficient*, *Jaccard Similarity* and *Dice Similarity*. However, the F-measure of these measures decreases when short strings are involved in matching, which limits their application range.

Levenshtein, *N Grams Distance*, and *Smith Waterman* are the most efficient similarity measures with high F-measure/runtime ratios, i.e. they produce the highest F-measures in polynomial time regardless of matching setups and datasets.

Conclusion Based on the previous observations, we can see different matching setups have influences not only on the F-measure but also the runtime. A similarity measure is only efficient when the most appropriate matching setup is used, i.e. a matching setup allowing minimizing the runtime while maximizing the F-measure of a similarity measure for a given dataset.

6.3. Guidelines

Based on the results of our experimental assessments, we describe some generic guidelines to make full use of similarity measures. It aims at facilitating the choices of similarity measures according to different criteria, namely matching setups, strings lengths and requirements on runtime. A heuristic algorithm is proposed below.

First, *Soundex*, *Jaro*, *Overlap Coefficient*, and *Euclidean Distance* are only suited for specific needs of matching [16, 11, 13], e.g. using *Soundex* to match homophones in English. In the context of *Unified Cube*s, the quality of the *correlative* mappings based on these similarity measures cannot be guaranteed due to their poor performance in matching instances from some generic sources. Therefore, they are not considered in the guidelines.

Second, in the case where only the Concatenated&Unprocessed matching setup is applicable (e.g. matching carried out by a non-expert user without specific knowledge of the sources), *N Grams Distance*, *Levenshtein*, *Smith Waterman*, and *Smith Waterman Gotoh* are the similarity measures allowing to obtain a high F-measure. In the case where descriptive information of instances can be matched in a *Seperated* and/or *Optimized* ways, the above-mentioned list of similarity measures can be complemented by *Monge Elkan*, *Needleman Wunch*, and *Jaro Winkler*.

Third, in the case where attribute instances contain relatively short strings (about 200 characters or less), *N Grams Distance*, *Levenshtein*, *Smith Waterman*, *Smith Waterman Gotoh*, *Monge Elkan*, *Needleman Wunch*, and *Jaro Winkler* are possible choices of similarity measures which can produce a high F-measure. In the case where attribute instances include long strings (more than 200 characters), all 12 similarity measures listed in table 5 allow producing a high F-measure.

At last, if the matching should be carried out within a short runtime, good choices of similarity measures would be *N Grams Distance*, *Levenshtein*, *Smith Waterman*, *Cosine Similarity*, *Block Distance*, *Matching Coefficient*,*Jaccard Similarity*, *Dice Similarity*, and *Needleman Wunch*.

As shown in figure 21, three similarity measures meet all criteria: *N Grams Distance*, *Levenshtein* and *Smith Waterman*. The high efficiency in various matching scenario makes them the most suitable string similarity measures to establish *correlative* mappings in a *Unified Cube*.

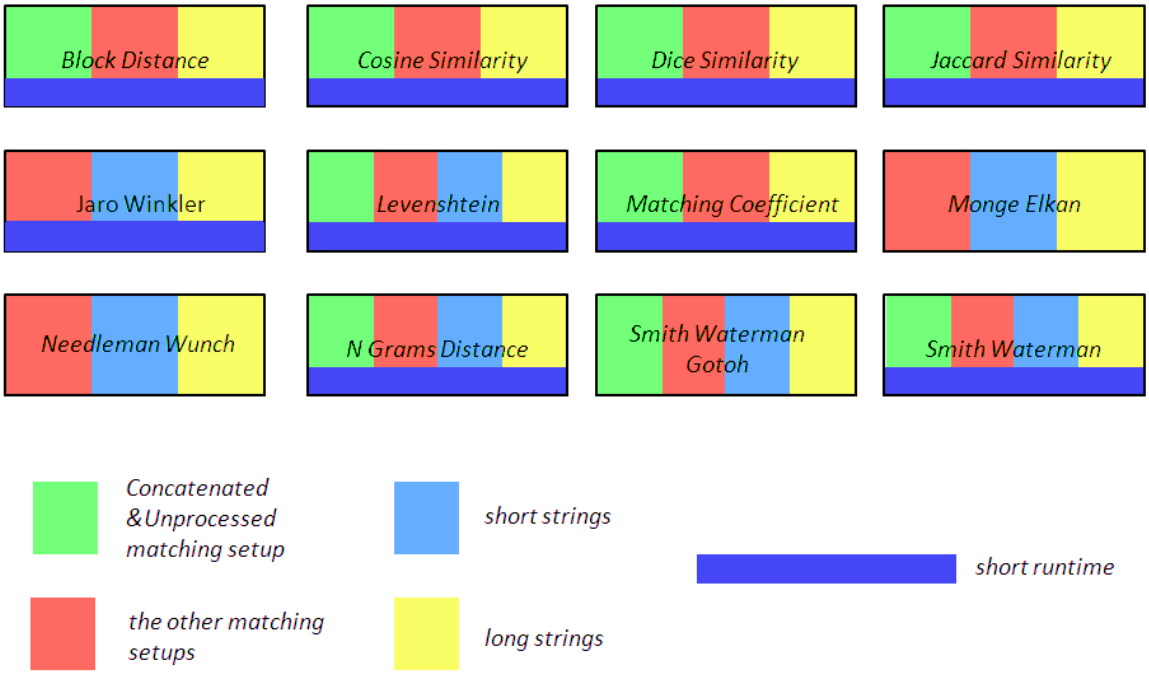


Figure 21: Similarity measures classified according to selection criteria

6.4. Validation

To validate our proposed guidelines, we use the benchmark of the *instance matching* track in the *Ontology Alignment Evaluation Initiative* campaigns 2016[20] and 2017[21]. The benchmark is named *False Positive Trap*. It includes two real-world datasets describing *musical works* in the catalogs coming from two French cultural institutions: *La Bibliothque Nationale de France* (BnF) and *La Philharmonie de Paris* (PP). The objective is to find equivalent instances of the *Music Work* classes scattered in two datasets of the benchmark. Each music work is associated with highly similar descriptions, such as title, compositors, etc. The *baselines* are provided along the benchmark. The benchmark of 2016 includes less than 3000 triples with an average string length of 61 characters in both datasets, while the benchmark of 2017 contains 6347 and 8622 triples with an average string length of 61 characters in both datasets.

To illustrate the genericity of our proposed guideline, it is necessary to carry out some tests between warehoused data and LOD. To do so, we build a DW in third normal form for the BnF dataset of both benchmarks. The PP dataset keeps its original RDF format.

Due to the large number of highly similar music works, we have to firstly write queries to extract corresponding descriptions of music work, for instance title versus title, compositors versus compositors, etc. Then, we use the Separated&Unprocessed matching setup to separately compare corresponding descriptions. According to our proposed guidelines, all similarity measures are suitable for the chosen matching setups. Moreover, the average string length of music work descriptions is rel-

[20] http://islab.di.unimi.it/content/im_oaei/2016/
[21] http://islab.di.unimi.it/content/im_oaei/2017/

atively short in the benchmarks. Consequently, the choices of string similarity measures are limited to those which work well with short strings. At last, to avoid excessively long runtime, a timeout is configured by the benchmark. Accordingly, the choices of similarity measures are restricted to those with a short runtime.

It leaves us four choices of similarity measures which are possibly appropriate for the matching in the benchmark: *Jaro Winkler*, *Levenshtein*, *N Grams Distance*, and *Smith Waterman*. We choose the *N Grams Distance* similarity measure due to its outstanding performance in our previous experimental assessments. The stable marriage algorithm is used to determine the correspondences among instances. As no semantic-based technique is used to improve the matching result, we expect some low similarity score between instances. Therefore, no threshold is defined for the matching. After the execution, we obtain some surprisingly good results. For the benchmark of 2016, we obtain a F-measure of 1. For the benchmark of 2017, we obtain a F-measure of 0.95. Hence, we can safely conclude that our matching process provides a feasible and efficient solution to generic instance matching problems.

7. Conclusion

Our aim is to make full use of as much information as possible to support effective and well-informed decisions. To this end, we have defined a generic conceptual multidimensional model, named *Unified Cube*, which blends data from multiple sources together. A conceptual *Unified Cube* unifies warehoused data and LOD in a business-oriented representation. The *Unified Cube* modeling breaks through three obstacles in the multidimensional modeling field: (i) warehoused data and LOD can be queried on-the-fly during analyses through *extraction formulae* of measures and attributes, (ii) a measure can be linked to a dimension starting from any level through *level-measure* mapping and (iii) attribute instances from one source are linked with equivalent instances of attribute from another source through *correlative* mappings.

We have proposed an implementation framework which manages interactions between a *Unified Cube* and multiple data sources at both the schema and the instance levels. Specifically, the framework (i) include a metamodel and an instantiation algorithm to provide a uniform manner to access different sources involved in a *Unified Cube* and (ii) contains a table of correspondences and an instance matching process to bridge the differences among multiple data sources at the instances level.

Based on our proposed implementation framework, we have designed an analysis processing process which enables analyses on multisource data in a user-friendly way. During analyses, decision-makers only need to interact with some business-oriented concepts in a *Unified Cube* without worrying about data sources. Our proposed analysis processing process automatically (i) generates queries to extract data from sources and (ii) merge extracted data together to prepare a dashboard.

We have defined a process which completes a *Unified Cube* schema by matching correlative instances from different sources. This process is applied to several real-world datasets to find the best configurations allowing maximizing the efficiency. The results of our experimental assessments have been integrated into some generic guidelines allowing identifying the most appropriate string similarity measures according to matching setup, string length, and requirement on runtime. We have validated the guidelines through benchmarks of a well-known matching evaluation campaign.

In the near future, we intend to study the efficiency of analyses on *Unified Cube*s built from large-scale data. We plan to combine *Unified Cube* with on-demand ETL techniques [4] to materialize up-to-date data according to decision-makers' analysis needs at querying time. One long-term perspective consists of automatically synchronizing a *Unified Cube* with schema evolutions in the data sources. It requires (i) identifying suitable techniques for automatic detection of source evolutions in the context of *Unified Cube*s and (ii) designing a process which automatically updates a *Unified Cube* according to source evolutions. With regards to the maintenance of the table of correspondences, several update alternatives would be included in the process, such as periodically executing our proposed matching process (cf. figure 5), triggering an update after each evolution detected in sources [23, 14], or triggering an update in an on-demand manner to support right-time business analyses [41].

References

[1] Abelló A, Darmont J, Etcheverry L, Golfarelli, M., Mazón, J.-N., Naumann, F., Pedersen, T., Rizzi, S. B., Trujillo, J., Vassiliadis, P., Vossen, G.: Fusion Cubes: Towards Self-Service Business Intelligence, *International Journal of Data Warehousing and Mining*, 2013;**9**(2):66–88. ISSN:1548-3924, 1548-3932.

[2] Abelló A, Romero O, Pedersen TB, Berlanga R, Nebot V, Aramburu MJ, Simitsis A. Using Semantic Web Technologies for Exploratory OLAP: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, 2015;**27**(2):571–588. ISSN:1041-4347.

[3] Abiteboul S, Manolescu I, Rigaux P, Rousset MC, Senellart P. *Web data management*, Cambridge University Press, 2011, ISBN:1-139-50505-X.

[4] Baldacci L, Golfarelli M, Graziani S, Rizzi S. QETL: An approach to on-demand ETL from non-owned data sources, *Data & Knowledge Engineering*, 2017;**112**:17–37. URL `https://doi.org/10.1016/j.datak.2017.09.002`.

[5] Bhattacharya I, Getoor L. Collective Entity Resolution in Relational Data, *ACM Transactions on Knowledge Discovery from Data*, 2006;**1**(1):5–44, ISSN:15564681. doi:10.1145/1217299.1217304.

[6] Boussaid O, Darmont J, Bentayeb F, Loudcher S. Warehousing Complex Data from the Web, *International Journal of Web Engineering and Technology*, 2008;**4**(4):408–433. ISSN:1476-1289. doi:10.1504/IJWET.2008.019942.

[7] Brizan DG, Tansel AU. A Survey of Entity Resolution and Record Linkage Methodologies, *Communications of the IIMA*, 2006;**6**(3):5–15.

[8] Castano S, Ferrara A, Montanelli S, Varese G. Ontology and Instance Matching, in: *Knowledge-Driven Multimedia Information Extraction and Ontology Evolution*, vol. 6050, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011 pp. 167–195. ISBN:978-3-642-20794-5 978-3-642-20795-2.

[9] Chaudhuri S, Dayal U, Narasayya V. An Overview of Business Intelligence Technology, *Communications of the ACM*, 2011;**54**(8):88–98, ISSN:00010782. doi:10.1145/1978542.1978562.

[10] Cheatham M, Hitzler P. String Similarity Metrics for Ontology Alignment, in: *The Semantic Web ISWC 2013*, vol. 8219, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013 pp. 294–309. ISBN:978-3-642-41337-7.

[11] Christian P. Soundex-can It Be Improved?, *Computers in Genealogy*, 1998;**6**:215–221. `http://www.essex.ac.uk/AMS/articles/Soundex.html`.

[12] Christophides V, Efthymiou V, Stefanidis K. Entity Resolution in the Web of Data, *Synthesis Lectures on the Semantic Web: Theory and Technology*, 2015;**5**(3):1–122, ISSN:2160-4711, 2160-472X. URL `https://doi.org/10.2200/S00655ED1V01Y201507WBE013`.

[13] Cohen W, Ravikumar P, Fienberg S. A Comparison of String Metrics for Matching Names and Records, *Kdd workshop on data cleaning and object consolidation*, 3, 2003.

[14] Curino C, Moon HJ, Deutsch A, Zaniolo C. Automating the Database Schema Evolution Process, *The VLDB Journal*, 2013;**22**(1):73–98, ISSN:1066-8888. doi:10.1007/s00778-012-0302-x.

[15] Deb Nath RP, Hose K, Pedersen TB. Towards a Programmable Semantic Extract-Transform-Load Framework for Semantic Data Warehouses, ACM Press, 2015, ISBN:978-1-4503-3785-4. doi:10.1145/2811222.2811229.

[16] Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures, *Proceedings of the VLDB Endowment*, 2008;**1**(2):1542–1552, ISSN:2150-8097. doi:10.14778/1454159.1454226.

[17] Etcheverry L, Vaisman A, Zimányi E. Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP, in: *Data Warehousing and Knowledge Discovery*, vol. 8646, Springer International Publishing, Cham, 2014 pp. 45–56. ISBN:978-3-319-10160-6.

[18] Euzenat J. *Ontology matching*, 2nd edition edition, Springer, New York, 2013, ISBN:978-3-642-38721-0.

[19] Getoor L, Machanavajjhala A. Entity Resolution: Theory, Practice & Open Challenges, *Proceedings of the VLDB Endowment*, 2012;**5**(12):2018–2019. ISSN:21508097.

[20] Ghawi R, Cullot N. Database-to-ontology mapping generation for semantic interoperability, *Third International Workshop on Database Interoperability (InterDB 2007)*, 91, 2007.

[21] Golfarelli M, Maio D, Rizzi S. Conceptual Design of Data Warehouses from E/R Schemes, *Thirty-First Annual Hawaii International Conference on System Sciences*, 7, IEEE Computer Society, Kohala Coast, HI, 1998, ISBN:978-0-8186-8255-1.

[22] Gusfield D, Irving RW. *The Stable Marriage Problem: Structure and Algorithms*, Foundations of Computing, MIT Press, Cambridge, Mass, 1989, ISBN:978-0-262-07118-5.

[23] Haase P, van Harmelen F, Huang, Z, Stuckenschmidt, H, Sure, Y.: *A Framework for Handling Inconsistency in Changing Ontologies*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005 pp. 353–367. ISBN:978-3-540-32082-1.

[24] Ibragimov D, Hose K, Pedersen TB, Zimányi E. Towards Exploratory OLAP over Linked Open DataA Case Study, HangZhou, 2014. doi:10.1007/978-3-662-46839-5_8.

[25] Kämpgen B, Harth A. Transforming Statistical Linked Data for Use in OLAP systems, *Proceedings of the 7th international conference on Semantic systems*, ACM Press, Graz, Austria, 2011, ISBN:978-1-4503-0621-8.

[26] Kämpgen B, ORiain S, Harth A. Interacting with Statistical Linked Data via OLAP Operations, *The Semantic Web: ESWC 2012 Satellite Events: ESWC 2012 Satellite Events*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015 pp.87–101. ISBN:978-3-662-46641-4. doi:10.1007/978-3-662-46641-4_7.

[27] Koudas N, Marathe A, Srivastava D. Flexible String Matching Against Large Databases in Practice, in: *Proceedings 2004 VLDB Conference*, Elsevier, 2004 pp. 1078–1086. ISBN:978-0-12-088469-8.

[28] Matei A, Chao KM, Godwin N. OLAP for Multidimensional Semantic Web Databases, in: *Enabling Real-Time Business Intelligence*, vol. 206, Springer Berlin Heidelberg, 2015 pp. 81–96. doi:10.1007/978-3-662-46839-5_6.

[29] Nebot V, Berlanga R, Pérez JM, Aramburu MJ, Pedersen TB. Multidimensional Integrated Ontologies: A Framework for Designing Semantic Data Warehouses, in: *Journal on Data Semantics XIII*, vol. 5530, Springer Berlin Heidelberg, 2009 pp. 1–36. doi:10.1007/978-3-642-03098-7_1.

[30] Pearson WR. Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA Algorithms, *Genomics*, 1991;**11**(3):635–650, ISSN:08887543.

[31] Pietriga E, Bizer C, Karger D, Lee R. Fresnel: A Browser-Independent Presentation Vocabulary for RDF, in: *The Semantic Web - ISWC 2006*, vol. 4273, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006 pp. 158–171, ISBN:978-3-540-49029-6 978-3-540-49055-5.

[32] Ravat F, Song J. Unifying Warehoused Data with Linked Open Data: A Conceptual Modeling Solution, *Model and Data Engineering (MEDI 2016)*, 9893, Springer International Publishing, Almeria, Spain, September 2016 pp. 245–259. doi:10.1007/978-3-319-45547-1_20.

[33] Ravat F, Song J, Teste O. Designing Multidimensional Cubes from Warehoused Data and Linked Open Data, *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS 2016)*, IEEE, Grenoble, France, 2016, ISBN:978-1-4799-8710-8. URL `http://dx.doi.org/10.1109/RCIS.2016.7549337`.

[34] Ravat F, Song J, Teste O. OLAP Analysis Operators for Multi-state Data Warehouses, *International Journal of Data Warehousing and Mining*, 2016;**12**(4):54–74, ISSN:1548-3924, 1548-3932. doi:10.4018/IJDWM.2016100102.

[35] Ravat F, Teste O, Tournier R, Zurfluh G. Algebraic and Graphic Languages for OLAP Manipulations, *International Journal of Data Warehousing and Mining*, 2008;**4**(1):17–46. URL `http://www.igi-global.com`.

[36] Romero O, Abelló A. Automating Multidimensional Design from Ontologies, *international workshop on Data warehousing and OLAP*, ACM Press, 2007. doi:10.1145/1317331.1317333.

[37] Saad R, Teste O, Trojahn C. OLAP Manipulations on RDF Data following a Constellation Model, *Proceedings of International Workshop on Semantic Statistics (SemStats 2013)* collocated with International Semantic Web Conference (ISWC-2013), Sydney, 2013.

[38] Salem R, Boussaïd O, Darmont J. Active XML-based Web data integration, *Information Systems Frontiers*, 2013;**15**(3):371–398, ISSN:1572-9419. doi:10.1007/s10796-012-9405-6.

[39] Trujillo J, Maté A. Business Intelligence 2.0: A General Overview, in: *Business Intelligence*, vol. 96, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012 pp. 98–116. doi:10.1007/978-3-642-27358-2_5.

[40] Ukkonen E. Approximate String-matching with q-grams and Maximal Matches, *Theoretical computer science*, 1992;**92**(1):191–211. ISSN:0304-3975. doi:10.1016/0304-3975(92)90143-4.

[41] Waas F, Wrembel R, Freudenreich T, Thiele M, Koncilia C, Furtado P. On-Demand ELT Architecture for Right-Time BI: Extending the Vision, *International Journal of Data Warehousing and Mining*, 2013;**9**(2):21–38. ISSN:1548-3924, 1548-3932. doi:10.4018/jdwm.2013040102.

[42] Wang P, Wu B, Wang B. TSMH Graph Cube: A novel framework for large scale multi-dimensional network analysis, IEEE, 2015. ISBN:978-1-4673-8272-4. doi:10.1109/DSAA.2015.7344826.

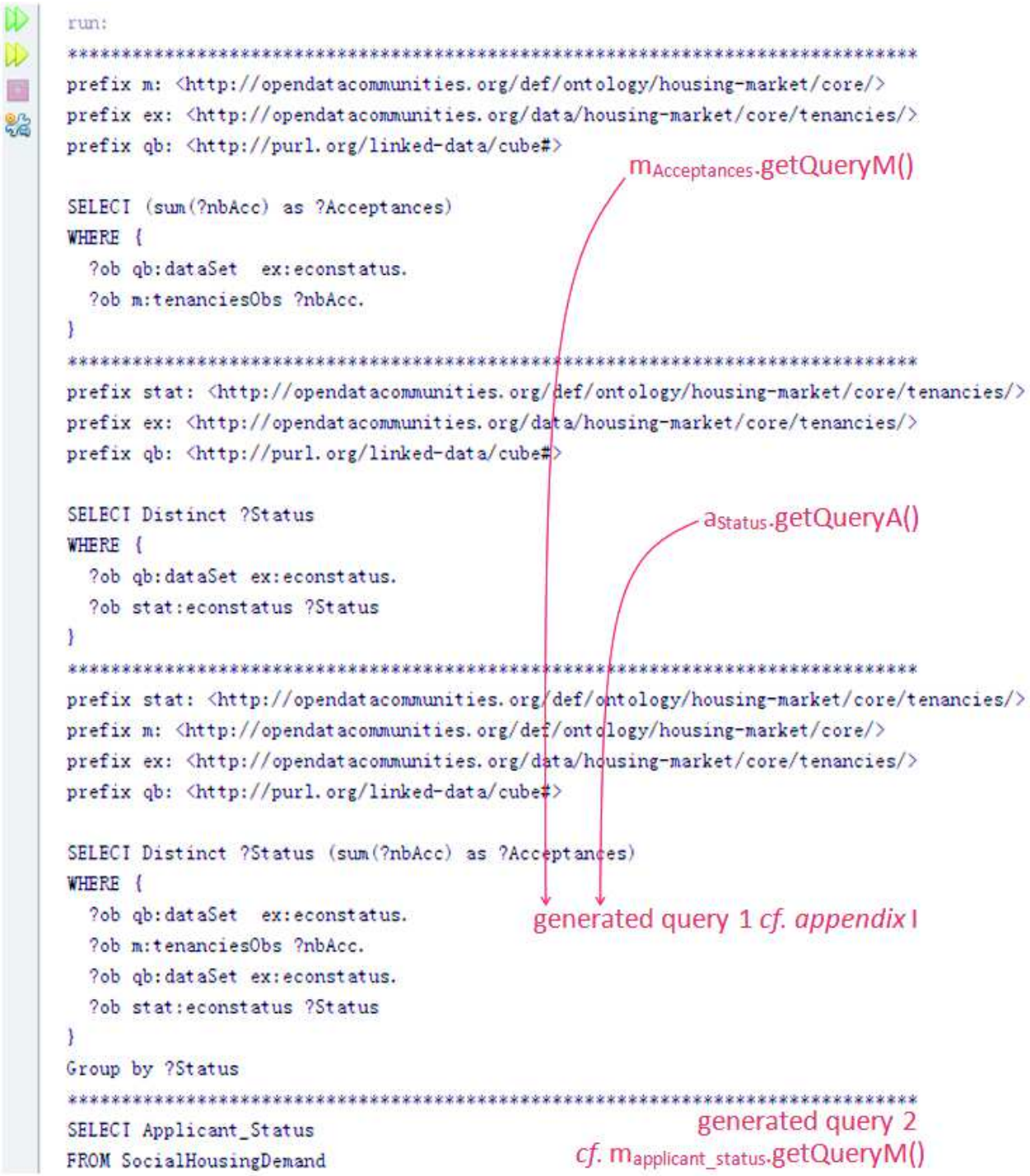
[43] Wilder-James E. Breaking Down Data Silos, *Harvard Business Review*, 2016.

[44] Yujian L, Bo L. A Normalized Levenshtein Distance Metric, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007;**29**(6):1091–1095, ISSN 0162-8828. doi:10.1109/TPAMI.2007.1078.

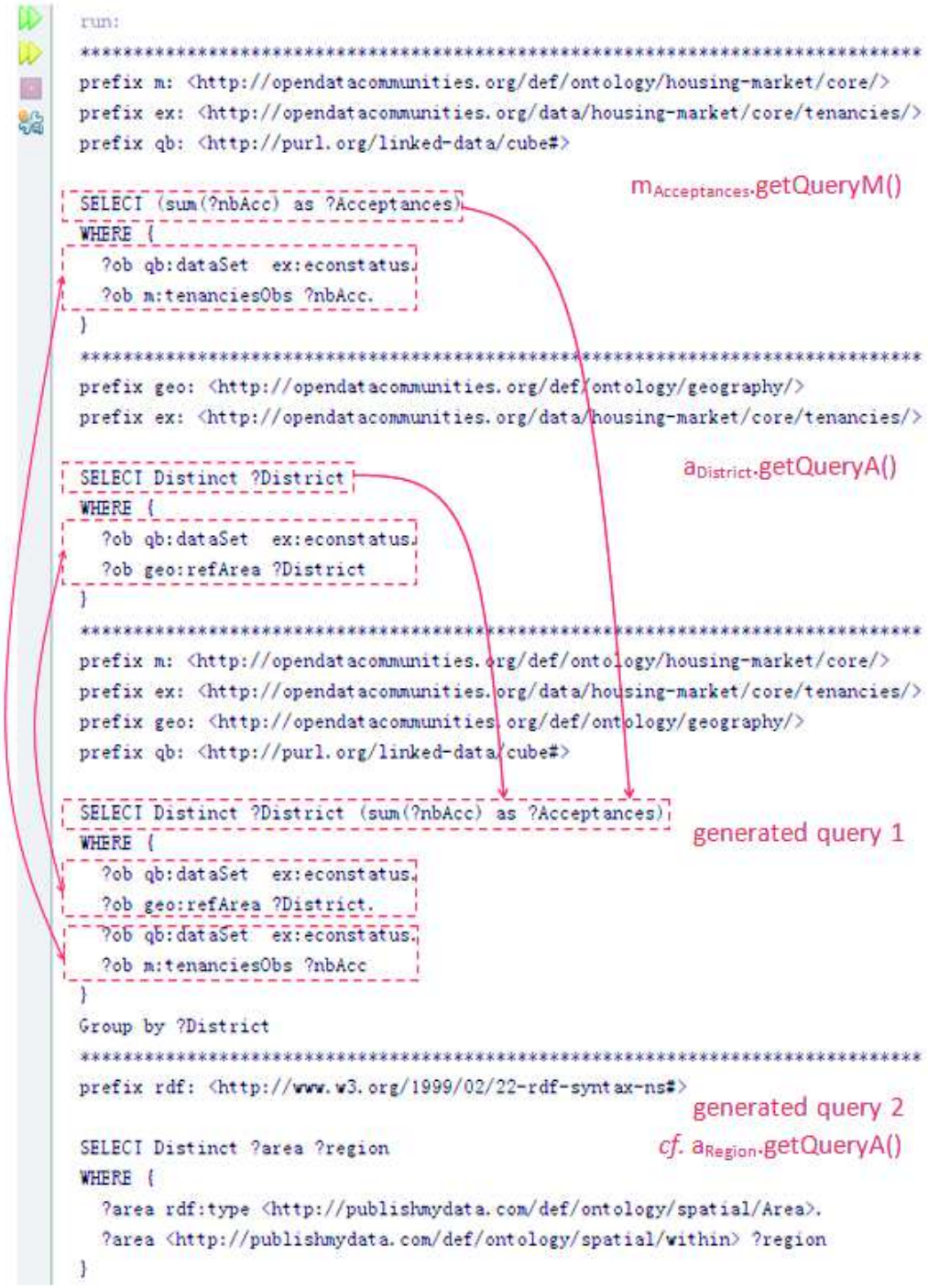
[45] Zorrilla ME, Mazón JN, Ferrández O, Garrigós I, Daniel F, Trujillo J. Eds. *Business Intelligence Applications and the Web: Models, Systems and Technologies*, IGI Global, 2012, ISBN:978-1-61350-038-5 978-1-61350-039-2.

A. Generated query of the second analysis

B. Generated queries of the third analysis



C. Generated queries of the fourth analysis



D. Generated queries of the fifth analysis